Peridigm

Peridigm Users Guide

For Peridigm versions $\geq 1.4.1$

Martin Rädel

DLR Deutsches Zentrum für Luft- und Raumfahrt
German Aerospace Center

DLR German Aerospace Center
Composite Structures and Adaptive Systems
Structural Mechanics
Dr. Tobias Wille

38108 Braunschweig
Germany

Tel: +49 (0)531 295-3701
Fax: +49 (0)531 295-3702
Web: http://www.dlr.de/fa/en

Martin Rädel
Tel: +49 (0)531 295-2048
Fax: +49 (0)531 295-2232
Mail: martin.raedel@dlr.de

Repository
This document is part of the PeriDoX repository [1].
The complete repository can be found at:
https://github.com/PeriDoX/PeriDoX

Citing
When citing this document, please reference the following:
Martin Rädel, Christian Willberg, Peridigm Users Guide, DLR-IB-FA-BS-2018-23, DLR Report, 2018, in PeriDoX, DOI: 10.5281/zenodo.1403015

Disclaimer
The contents of this document are provided "AS IS". This information could contain technical inaccuracies, typographical errors and out-of-date information. This document may be updated or changed without notice at any time. Use of the information is therefore at your own risk. In no event shall the DLR be liable for special, indirect, incidental or consequential damages resulting from or related to the use of this document.

Contents

List of Figures

List of Tables

List of Symbols

Scalars

Symbol	Name	Description
a	Acceleration	
c	Bond constant	
h	Thickness	
s	Stretch	
t	Time	
Δt	Time step	
u	Displacement	
v	Velocity	
x	x axis	
y	y axis	
z	z axis	
A	Surface area	
E	Young's modulus	
F	Force	
G	Shear modulus	
G_{0C}	Critical energy release rate	
G_I	Energy release rate mode I	
G_{IC}	Critical energy release rate mode I	
G_{II}	Energy release rate mode II	
G_{IIC}	Critical energy release rate mode II	
K	Bulk modulus	
V	Volume	
δ	Horizon	
ε	Strain	
γ	Shear strain	Engineering strain
ν	Poisson ratio	
$\omega_{\boldsymbol{\xi}}$	Radial influence function	
ρ	Density	
σ	Stress	
τ	Shear stress	Engineering stress

Symbol	Name	Description

Vectors

Symbol	Name	Description
$\boldsymbol{\xi}$	Bond vector, undeformed	Vector from a point $\mathbf{x}$ in the family of $\mathbf{x}$ to $\mathbf{x}'$

Indices

Symbol	Description
$(\ )_C$	Critical
$(\ )_H$	Hardening
$(\ )_I$	Fracture mode I
$(\ )_{II}$	Fracture mode II
$(\ )_{\mathrm{y}}$	Yield

Operators

Symbol	Description
$\Delta(\ )$	Difference

Acronyms

BSD	Berkeley Software Distribution
GPL	GNU General Public License
LPS	linear peridynamic solid

1. About

1.1. Motivation

> *"In peridynamics, cracks are part of the solution, not part of the problem."*
> — Florin Bobaru

1.2. Scope

This document is supposed to be the users guide for the creation of models, performing simulations with *Peridigm* and postprocessing with *ParaView*. It serves as an addition to the original and official user guide [2]. The open-source version of this document is based upon [3].

The document includes a growing quick reference guide to model options in *Peridigm* as well as conclusions to obtain proper results. Additionally, preferences and possibilities for the postprocessing of the simulation results with *ParaView* are part of the document.

All informations in this document have been collected to the best of our knowledge from application of or coding in *Peridigm*.

1.3. Contributors

This document represents a cumulative effort. The following list shows the contributors in alphabetic order by last name.

- Anna-Janina Bednarek
- Martin Rädel
- Lasse Wiedemann
- Christian Willberg

1.4. Peridynamic codes

Peridigm is one of several codes implementing peridynamics. Following are lists of known codes dealing with the numerical implementation of the peridynamic theory.

1.4.1. Production codes

Table 1.1.: Codes implementing peridynamics

Name	Open-Source	License	Language				Source
			C	C++	Fortran	Python	
EMU	-	-	-	-	✓	-	
LAMMPS	✓	GPL	-	✓	-	-	[4]
PDpy	✓	-	-	-		✓	
Peridigm	✓	BSD	-	✓	-	-	[2]
PeriFlakes	✓	GPL	✓	-	-	✓	
PeriPyDIC	✓	GPL	C	-	-	✓	
Sierra/Solid Mechanics (Presto)	-						[5]

1.4.2. Others

- https://github.com/matteopolleschi/Mixed-implicit-explicit-peridynamic-code

1.4.3. Groups with unpublished peridynamic codes

- PMMA Research Lab, Dr. Oterkus — University of Strathclyde, Glasgow
- AAML Group, Prof. Hong — KAIST, Korea
- m^3 Group, Prof. Madenci — University of Arizona, USA
- David Miranda — University of Bath, UK

2. *Peridigm* - Basics

2.1. Preliminaries

2.1.1. Typographical conventions

All typographical conventions are found in Table 2.1.

Table 2.1.: Notational conventions

Symbol	General	Description
a,α	minuscule	Scalar in $\mathbb{R}$, Tensor of rank 0
$\mathbf{a}$, $\boldsymbol{\alpha}$	bold minuscule	Vector in $\mathbb{R}^3$, Tensor of rank 1
$\mathbf{A}$	bold majuscule	Matrix in $\mathbb{R}^{n\times m}$, Dyad, Tensor of rank 2
$\mathbf{A}$ with A_{ijk}, A_{ijkl}	bold majuscule	Triade, Tetrade, Tensor of rank > 2
$\underline{a}$	minuscule, underline	Scalar state
$\underline{\mathbf{A}}$	bold majuscule, underline	Vector state
$\underline{\mathbb{A}}$	blackboard majuscule, underline	Double state

2.1.2. Coordinate systems

Peridigm uses a global cartesian right-hand coordinate system. Local coordinate systems are currently not available.

2.1.3. Unit systems

For the application of all methods, tools and solutions input values are required. It has to be made sure by the user to make sure that all input values are defined in a single consistent unit system, like the SI- or imperial unit system. The following Table 2.2 shows some well-established unit systems.

Table 2.2.: Consistent unit systems according to *ANSYS* /UNITS command

Value	Unit system				
	SI	CGS	MPA	BFT	BIN
Mass	[kg]	[g]	[t]	[slug]	$\frac{[\text{lbf}][\text{s}]^2}{[\text{in}]}$
Length	[m]	[cm]	[mm]	[ft]	[in]
Time	[s]	[s]	[s]	[s]	[s]
Temperature	[K]	[K]	[K]	[°R]	[°R]
Velocity	$\frac{[\text{m}]}{[\text{s}]}$	$\frac{[\text{cm}]}{[\text{s}]}$	$\frac{[\text{mm}]}{[\text{s}]}$	$\frac{[\text{ft}]}{[\text{s}]}$	$\frac{[\text{in}]}{[\text{s}]}$
Acceleration	$\frac{[\text{m}]}{[\text{s}]^2}$	$\frac{[\text{cm}]}{[\text{s}]^2}$	$\frac{[\text{mm}]}{[\text{s}]^2}$	$\frac{[\text{ft}]}{[\text{s}]^2}$	$\frac{[\text{in}]}{[\text{s}]^2}$
Force	[N]	[dyn]	[N]	[lbf]	[lbf]
Moment	[N] [m]	[dyn] [cm]	[N] [mm]	[lbf] [ft]	[lbf] [in]
Pressure	[Pa]	[Ba]	[MPa]	[lbf] / [ft]^2	[psi]
Density	$\frac{[\text{kg}]}{[\text{m}]^3}$	$\frac{[\text{g}]}{[\text{cm}]^3}$	$\frac{[\text{t}]}{[\text{mm}]^3}$	$\frac{[\text{slug}]}{[\text{ft}]^3}$	$\frac{[\text{lbf}][\text{s}]^2/[\text{in}]}{[\text{in}]^3}$
Energy	[J]	[erg]	[mJ]	[ft] [lbf]	[in] [lbf]
Energy release rate	$\frac{[\text{J}]}{[\text{m}]^2} = \frac{[\text{N}]}{[\text{m}]}$	$\frac{[\text{erg}]}{[\text{cm}]^2}$	$\frac{[\text{mJ}]}{[\text{mm}]^2} = \frac{[\text{N}]}{[\text{mm}]}$		

2.1.4. Conversion between peridynamic and finite element values

Material properties

- 3D — Without plane stress or strain assumptions
- 2D — With plane stress or strain assumptions

Note, that in case of bond-based analysis, the conversion is problem-dependent:

$$\nu = \begin{cases} \frac{1}{4} & \text{3D \& 2D plane strain} \\ \frac{1}{3} & \text{2D plane stress} \end{cases} \tag{2.1}$$

Table 2.3.: Material property conversion

Description	Dim.	Equation	Source
Bulk modulus	3D	$K_{3D} = \frac{E}{3(1-2\nu)}$	
	2D	$K_{2D} = \begin{cases} \frac{E}{2(1-\nu)} & \text{, plane stress} \\ \frac{E}{2(1-\nu-2\nu^2)} & \text{, plane strain} \end{cases}$	[6]
	1D	$K_{1D} = E$	
Shear modulus	all	$G = \frac{E}{2(1+\nu)}$	
Bond constant - bond based, constant influence function			
	3D	$c_{3D} = \frac{18K}{\pi\delta^4} = \frac{30G}{\pi\delta^4}$	[6–8][1]
	2D	$c_{2D} = \frac{12K}{\pi h\delta^3} = \frac{G}{\pi h\delta^3} \cdot \begin{cases} 20 & \text{plane strain} \\ 24 & \text{plane stress} \end{cases}$	[6]
	1D	$c_{1D} = \frac{2K}{A\delta^2}$	[6]
Critical Stretch - bond based			[9][2]
	3D	$s_{C}3\text{D} = \sqrt{\frac{5G_{0C}}{9K_{3D}\delta}} = \sqrt{\frac{10G_{0C}}{\pi c_{3D}\delta^5}}$	[6, 7]
	2D	$s_{C}2\text{D} = \sqrt{\frac{\pi G_{0C}}{3K_{2D}\delta}} = \sqrt{\frac{4G_{0C}}{c_{2D}h\delta^4}}$	[6]
	1D	$s_{C}1\text{D} = \sqrt{\frac{3G_{0C}}{K_{1D}\delta}} = \sqrt{\frac{6G_{0C}}{c_{1D}A\delta^3}}$	[6]
Critical Stretch - state based			
	3D	$s_{C}3\text{D} = \sqrt{\frac{G_{0C}}{\left[3G + \left(\frac{3}{4}\right)^4 \left(K - \frac{5G}{3}\right)\right]\delta}}$	[8, 10][3]
	2D	$s_{C}2\text{D} = \sqrt{\frac{G_{0C}}{\left[\frac{6}{\pi}G + \frac{16}{9\pi^2}(K - 2G)\right]\delta}}$	[8, 10]
	1D	$s_{C}1\text{D} =$	

Internally, Peridigm uses K, G for the stiffness description. The conversion between

[1] [6, p.30], [7, p.1529], [8, p.37]
[2] [9, p.114]
[3] [8, p.120]

engineering constants and K, G is performed in methods

- `PeridigmNS::Material::calculateBulkModulus`
- `PeridigmNS::Material::calculateShearModulus`

in class `/src/Materials/Peridigm_Material.cpp`.

Currently, Peridigm only uses the 3D formulation of the peridynamics, e.g. for surface correction factors. Therefore, only the 3D equations shall be used for material modelling.

2.2. Model input file format

Peridigm requires two input files: the model and the mesh. This section describes the format of the model input file, while section 2.3 is devoted to the possible types of discretization and the respective formats.

2.2.1. Input deck structure

This section is basically taken from section 3.2 in [2]. Independent of the actual input file format, a Peridigm input deck takes the form:

- `[Discretization Section]` required
- `[Materials Section]` required
- `[Damage Models Section]` optional
- `[Blocks Section]` required
- `[Contact Section]` optional
- `[Boundary Conditions Section]` required
- `[Compute Class Parameters Section]` optional
- `[Solver Section]` required
- `[Output Section]` required

We elaborate on each of these sections below:

Table 2.4.: Input file section overview

Section	Description
`[Discretization Section]`	Contains filename of input mesh, or arguments to Peridigm internal mesh generator.
`[Materials Section]`	Contains the names of the material models used and arguments for their constitutive parameters.
`[Damage Models Section]`	Contains the names of the damage models used and arguments for their constitutive parameters.
`[Blocks Section]`	Contains a listing of the model properties, associating each block with material and damage models as well as a horizon.
`[Contact Section]`	Contains a listing of the contact models and the kind of contact model to be applied when model regions come into contact in a simulation.

continued ...

...continued

Section	Description
`[Boundary Conditions Section]`	Contains the initial and boundary conditions and loads for the simulation.
`[Compute Class Parameters Section]`	Contains user defined calculation data which shall be output in a simulation.
`[Solver Section]`	Contains the solver to be used along with solver parameters.
`[Output Section]`	Contains the output filename and output frequency along with a listing of the variables to be output.

Each of these sections is described in a separate part of chapter 3.

2.2.2. Formats

Peridigm currently supports three different input file formats:

- XML format
- Peridigm/Free format
- YAML format

Both, XML and free format are more or less obsolete and will not be part of future developments. For version 1.5 and beyond the focus is on the YAML format.

2.2.2.1. XML format

Comment sign

A comment line or block is started with `<!--` and ends with `-->`

Single-line comment

```
<!-------Your comment----->
```

Multi-line comment example

from a generic xml file

```
<detail>
  <band height="20">
  <!--
    Hello,
       I am a multi-line XML comment
    <staticText>
      <reportElement x="180" y="0" width="200" height="20"/>
      <text><![CDATA[Hello World!]]></text>
    </staticText>
  -->
  </band>
</detail>
```

Format interference with the function parser

When you write a xml format document, you cannot use “>” or “<” directly. When you edit a xml document, you should use “>” replace “>” and “<” replace “<”, e.g.

```
<Parameter name="Value" type="string" value="if(t &lt;= 10.0e-6)\{value
    =45.8216*t;\} else \{value=0;\}"/>
```

instead of

```
<Parameter name="Value" type="string" value="if(t <= 10.0e-6)\{value
    =45.8216*t;\} else \{ value=0;\}"/>
```

2.2.2.2. Free format

This format is no longer supported in the GitHub master.
For users that have existing .peridigm input decks, a utility for converting them to .yaml is available at
`scripts/peridigm_to_yaml.py`
This script was used for conversion of all the .peridigm files in the repository, with the exception of twist_and_pull.peridigm, which required manual editing to remove aprepro commands. Aprepro is not supported for .yaml or .xml input decks, and all aprepro support will go away when support for .peridigm files is removed.

Comment sign

A comment line is started with the hash sign: #.

2.2.2.3. YAML format

Build requirements

Note that using the new .yaml input files requires that Trilinos be built with YAML support. See the installation guide for instructions.

Comment sign

A comment line is started with the hash sign: #. Comments can start anywhere on a line, and continue until the end of the line. YAML does not support block comments.

2.2.3. Input file parser

The input is parsed directly in the `Peridigm.cpp` class in `src/core/`.

The input of keywords seems to be case-insensitive.

2.2.4. Function parser

2.2.4.1. Preliminaries

Description

You can use functions inside a *Peridigm* input deck. This means, it is possible to use algebraic expressions, e.g. for the definition of boundary conditions. Peridigm uses the RTCompiler function parser to process C-style expressions for the specification of input parameters, including initial and boundary conditions. The RTCompiler (RTC) is used via the Trilinos PG_RuntimeCompiler class.

History

The internal mechanism for processing functions used to be muParser, and was recently changed to rtcompiler, see this commit.

2.2.4.2. Supported operations

The RTC language can be thought of as a small subset of the C language with a couple minor modifications. The following information is repeated from the RTCompiler source code documentation.

Operators

The RTC language has the following operators that work exactly as they do in C and have the same precedence as they do in C:

- + Addition
- − Subtration
- − Negation
- ∗ Multiplication
- / Division
- ^ Exponentiation[4]
- == Equality
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to
- = Assignment
- || Logical or
- && Logical and
- != Inequality
- % Modulo
- ! Logical not

Control flow

The RTC language has the following control flow statements:

- for(expr ; expr ; expr) { ... }
- while(expr) { ... }
- if (expr) {...}
- else if (expr) {...}
- else {...}

These control flow statements work exactly as they do in C except that the code blocks following a control flow statement **MUST** be enclosed within braces even if the block only consists of one line.

Line Structure

The line structure in the RTC language is the same as that of C. Expressions end with a semicolon unless they are inside a control flow statement.

According to this commit the .peridigm input deck format (free format) does not support multi-line entries.

[4] This operator does not occur in the C language, but is added into the RTC language for convenience.

Math

The following math.h functions are available in RTC. Peridigm also seems to support the APREPRO functionalities. The features are reported in [11].

Table 2.5.: Math commands for Peridigm function parser

Syntax	Description
abs(x)	Absolute value of x
acos(x)	Inverse cosine of x, returns radians
acosd(x)	Inverse cosine of x, returns degrees
acosh(x)	Inverse hyperbolic cosine of x
asin(x)	Inverse sine of x, returns radians
asind(x)	Inverse sine of x, returns degrees
asinh(x)	Inverse hyperbolic sine of x
atan(x)	Inverse tangent of x, returns radians
atan2(x,y)	Inverse tangent of y/x, returns radians
atan2d(x,y)	Inverse tangent of y/x, returns degrees
atand(x)	Inverse tangent of x, returns degrees
atanh(x)	Inverse hyperbolic tangent of x
ceil(x)	Smallest integer not less than x
cos(x)	Cosine of x, with x in radians
cosd(x)	Cosine of x, with x in degrees
cosh(x)	Hyperbolic cosine of x
d2r(x)	Degrees to radians
dim(x,y)	$x - \min x, y$
dist(x1,y1,x2,y2)	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
erf(x)	Error function
erfc(x)	Complementary error function $(1.0 - \mathrm{erf}(x))$
exp(x)	Exponential e^x
fabs(x)	returns the absolute value of x
floor(x)	Largest integer not greater than x
fmod(x,y)	Floating-point remainder of x/y
gamma(x)	returns $\Gamma(x)$
hypot(x,y)	$\sqrt{x^2 + y^2}$
int(x), [x]	Integer part of x truncated toward 0
j0(x)	Bessel function of order zero
j1(x)	Bessel function of order one
i0(x)	Modified Bessel function of order zero
i1(x)	Modified Bessel function of order one
julday(mm,dd,yy)	Julian day corresponding to mm/dd/yy
juldayhms(mm,dd,yy,hh,mm,ss)	Julian day corresponding to mm/dd/yy at hh:mm:ss

continued ...

. . . continued

Syntax	Description
lgamma(x)	$\log(\Gamma(x))$
ln(x)	Natural (base e) logarithm of x
log(x)	Natural (base e) logarithm of x
log10(x)	Base 10 logarithm of x
log1p(x)	$\log(1+x)$ Accurate even for very small values of x
max(x,y)	Maximum of x and y
min(x,y)	Minimum of x and y
nint(x)	Rounds x to nearest integer. <0.5 down; >= 0.5 up
polarX(r,a)	$r \cdot \cos(a)$, a is in degrees
polarY(r,a)	$r \cdot \sin(a)$, a is in degrees
pow(b, e)	returns b to the e power (Note: you can use the Exponentiation operator instead)
r2d(x)	Radians to degrees
rand(xl,xh)	Random value between xl and xh; uniformly distributed
rand_lognormal(m,s)	Random value with lognormal distribution with mean m and stddev s
rand_normal(m,s)	Random value normally distributed with mean m and stddev s
rand_weibull(a, b)	Random value with weibull distribution with $\alpha = a$ and $\beta = b$
sign(x,y)	$x \cdot \text{sign}(y)$
sin(x)	Sine of x, with x in radians
sind(x)	Sine of x, with x in degrees
sinh(x)	Hyperbolic sine of x
sqrt(x)	Square root of x
srand(seed)	Seed the random number generator with the given integer value. At the beginning of Aprepro execution, srand() is called with the current time as the seed.
strtod(svar)	Returns a double-precision floating-point number equal to the value represented by the character string pointed to by svar.
tan(x)	Tangent of x, with x in radians
tand(x)	Tangent of x, with x in radians
tanh(x)	Hyperbolic tangent of x
Vangle(x1,y1,x2,y2)	Angle (radians) between vector $x_1\hat{i}+y_1\hat{j}$ and $x_2\hat{i}+y_2\hat{j}$
Vangled(x1,y1,x2,y2)	Angle (degrees) between vector $x_1\hat{i}+y_1\hat{j}$
word_count(svar,del)	Number of words in svar. Words are separated by one or more of the characters in the string variable del.

2.2.4.3. Variables

User variables

Definition

It is possible to define your own variables for use inside a Peridigm input deck. The variables have to be defined at the beginning of the input deck. For YAML-input this is before the line *Peridigm:*. The definition starts with a hash key, followed by the variable name, the equal sign and subsequently the variable value. It is possible to perform algebraic operations inside the variable definition.

```
#{LENGTH=1.0}
#{WIDTH=0.25}
#{NUM_ELEM_ALONG_WIDTH=5}
#{ELEMENT_SIZE=WIDTH/NUM_ELEM_ALONG_WIDTH}
#{HORIZON=0.15}
```

Usage

A variable value is used inside the model definition by calling the variable name in curly braces, e.g. for the variable *HORIZON* is the following example excerpt of a block definition in an input file:

```
Blocks:
  Block_1:
    Block Names: "Block_1"
    Material: "testmaterial"
    Horizon: {HORIZON}
```

Pre-defined variables

There are several pre-defined variables that can be used in the input deck. They are defined in `/src/core/Peridigm_BoundaryCondition.cpp`

The current list contains:

Variable	Description
t	Time
x	x-position of a peridynamic collocation point in the global cartesian coordinate system at time t
y	y-position of a peridynamic collocation point in the global cartesian coordinate system at time t
z	z-position of a peridynamic collocation point in the global cartesian coordinate system at time t
value	Prefix for string keyword entries

2.2.4.4. List of examples

↗ `/examples/twist_and_pull/twist_and_pull.peridigm`

2.3. Discretization input file format

2.3.1. Preliminaries

The mesh-free discretization for Peridigm consists of peridynamic collocation points with coordinates in three-dimensional space and an associated volume.

The discretization can either be an Albany multiphysics or Exodus finite element mesh or the peridynamic collocation points can be input directly. In case of the Albany or Exodus mesh description, the conversion from the finite element mesh to the peridynamic collocation points is performed internally by Peridigm for a number of supported element types. The supported element types are explained in the respective sections.

The discretization is generally stored in an external file and input into the Peridigm model file by referencing the discretization file name.

2.3.2. Text File – Direct collocation point text file input

2.3.2.1. Description

Discretization based on the direct specification of peridynamic collocation points.

2.3.2.2. Mesh file layout

The ASCII mesh file is space-separated. It consists of the following five columns in this order:

Column	Value	Description
1	x	x-coordinate of the peridynamic collocation point in the global cartesian coordinate system
2	y	y-coordinate of the peridynamic collocation point in the global cartesian coordinate system
3	z	z-coordinate of the peridynamic collocation point in the global cartesian coordinate system
4	block_id	Block-ID for property assignment
5	volume	Collocation point volume

Following is an exemplary input file where all points belong to Block 1 and have the same volume 1.0.

```
# x y z block_id volume
-1.0  -0.5   0.5   1   1.0
-1.0  -0.5  -0.5   1   1.0
-1.0   0.5   0.5   1   1.0
-1.0   0.5  -0.5   1   1.0
...
```

2.3.2.3. Code

- from `src/io/discretization/`:
  - `Peridigm_TextFileDiscretization.cpp`
  - `Peridigm_TextFileDiscretization.hpp`

2.3.3. Exodus – finite element mesh

2.3.3.1. Description

Discretization based on a finite element mesh in binary *Exodus* format.

2.3.3.2. Mesh file layout

To get an idea how the binary exodus format is setup have a look at [12].

Convert binary Exodus mesh to ASCII text file

The Exodusformat uses the HDF5-library and especially the included NetCDF-C library. NetCDF-C provides tools to convert the binary Exodus format to a human-readable ASCII equivalent and vice-versa. NetCDF-Cis required to build Peridigm. Therefore, if Peridigm is available on your system, these tools are also available. In case Peridigm is not installed on your system, use the FETranslator User Guide from the FESuite-Repository for instructions on how to install the necessary tools on your operating system.

In case you want to convert the binary mesh in file `$FILE.g$` into a normal text file, open a command line and type

```
ncdump $FILE.g > $FILE.g.ascii
```

Afterwards, the file `$FILE.g.ascii` contains the text file.

Convert a ASCII text file into a binary Exodus mesh

In case you prepared an ASCII text file `$FILE.g.ascii` which is the equivalent of a binary Exodus mesh file, you can translate the text file to a binary mesh file `$FILE.g$` with

```
ncgen -o $FILE.g $FILE.g.ascii
```

2.3.3.3. Code

- `/src/io/discretization/Peridigm_ExodusDiscretization.cpp`
- `/src/io/discretization/Peridigm_ExodusDiscretization.hpp`

2.3.3.4. Possibilities to create format

Finite element meshes in the *Exodus* format can be created using

- *CUBIT* https://cubit.sandia.gov/
- gmsh → vtk → exodus
- gmsh-exodus-converter https://github.com/IaPCS/gmsh-exodus-converter

2.3.3.5. Supported element types

SPHERE_ELEMENT

From Sierra/SolidMechanics

TET_ELEMENT

Sketch: TO-DO

10-node tetrahedron elements are treated as 4-node tetrahedron elements. Middle nodes on the element edges are discarded.

Method for conversion: `tetCentroidAndVolume()` in `Peridigm_GeometryUtils.cpp`

HEX_ELEMENT

Sketch: TO-DO

20-node hexahedron elements are treated as 8-node hexahedron elements. Middle nodes on the element edges are discarded.

Method for conversion: `hexCentroidAndVolume()` in `Peridigm_GeometryUtils.cpp`

2.3.3.6. How the conversion between FE mesh to *Peridigm* collocation points works

Von: Littlewood, David John [mailto:djlittl@sandia.gov]
Gesendet: Donnerstag, 10. März 2016 17:46
An: Parks, Michael L; Rädel, Martin
Cc: peridigm-users@software.sandia.gov
Betreff: Mesh conversion

Martin,

As Mike said, Peridigm uses the approach of converting a hex or tet element into a single nodal volume. The routines in Peridigm are called tetCentroidAndVolume() and

hexCentroidAndVolume(), which are in Peridigm_GeometryUtils.*pp. The routine for converting a 4-noded tet is trivial. The conversion routine for a 8-noded hex operates by dividing the hex into 24 tets, using temporary points at the face barycenters and one at the element barycenter. Once we have the 24 tets, the conversion routines for tets are applied and the results appropriately summed together to get the hex centroid and volume. The tet routines are exact, and the hex routines are exact if the hex has planar faces. If the hex has non-planar faces, the routines result in a faceted approximation of the hex. The use of 24 hexes guarantees that the approach is not orientation dependent. It would be possible, I believe, to do something more complex using finite-element shape functions, but this involves making some assumptions (the code is given only node and connectivity information, information on shape functions is generally not included in mesh files).

For 10-noded tets and 20-noded hexes, Peridigm ignores all the "extra" nodes and considers only the 4 corner nodes for a tet and the 8 corner nodes for a hex.

One thing to be aware of is the need to transfer node sets from the original hex/tet mesh to the peridynamic discretization. The Peridigm algorithm includes the newly-created nodal volume in a node set if any of the nodes in the original element are in the node set. Peridigm ignores sides sets, as they are generally used for applying pressure/traction loads, which are not directly applicable to nonlocal models and/or meshfree discretizations.

The result of the conversion routine is a point cloud that defines (x, y, z, volume, block_id) for each node in the discretization. In the code, the block_id is used to associate a material model with the nodal volume, as well as some other operations.

One caution regarding converting pre-existing hex/tet meshes is that peridynamics is not well suited to handle highly graded meshes. The mesh does not need to be uniform, but if the mesh contains elements with, for example, 10-to-1 differences in element size, you'll run into problems. In this case, you'll have to choose a horizon that is several times larger than the large elements, leading to the situation where the horizon is many times larger than the smallest elements. In most cases this fails to achieve the high fidelity you were looking for in the highly-refined portion of the mesh, and it can really hurt code performance.

Your options for reading in pre-existing mesh are 1) convert the mesh to exodus/genesis format, in which case Peridigm can read it directly and takes care of everything, 2) pre-process the mesh yourself and create a text file with (x, y, z, volume, block_id) information and use Peridigm's text-file reader, or 3) implement a new discretization type that reads your mesh files into Peridigm. In the last case, you'd be able to leverage a lot of the code in the existing exodus file reader, but obviously the specifics of the file format would be different.

If it were me, I'd go with option 1), and I'd use the exodus.py python tool that is distributed as part of SEACAS. The SEACAS package is distributed with Trilinos, so you should have access to that already if you're running Peridigm. There's a learning

curve associated with the exodus format, but if you're able to convert to exodus format then you can use all the SEACAS utilities, for example decomp for creating parallel decompositions. (Side note, technically exodus is an output file format, and genesis is the corresponding input file format, but in practice people call both of them exodus.)

Hope this helps, Dave

2.3.4. Albany – multiphysics mesh

2.3.5. PdQuickGrid – Internal mesh generator

3. *Peridigm* - Quick reference guide

3.1. Discretization

3.1.1 Description

Specify the discretization. Different types of discretization are supported, see section 2.3.

3.1.2 Literature

- [7]

3.1.3 Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- from `src/io/discretization/`:
  - Basically all classes

3.1.4 Input parameters

List

Name	Type	Required	Default	Description
Type[1]	string	✓	-	Mesh key: "Exodus" | "Text File" | "Albany" | "PdQuickGrid"
Input Mesh File	string	✓	-	Name of the mesh file in the same folder
Influence Function[2]	string	-	"One"	"One" | "Parabolic Decay" | "Gaussian" | User defined
Omit Bonds Between Blocks[3]	string	-	-	"All" | "None" | List of blocks to ignore bonds between
Bond Filters[4]	List	-	-	List of Bond Filters

Remarks

1. Discretization type:

   Text File: ASCII text file mesh input, see 2.3.2
   Exodus: Binary Exodus mesh input, see 2.3.3
   Albany: Mesh from Albany multiphyisics code, see 2.3.4
   PdQuickGrid: Mesh creation inside *Peridigm* for simple cubical geometries, see 2.3.5

2. See section 3.2.3
3. Possible values for keyword *Omit Bonds Between Blocks*:
   - "All"
   - "None"
   - Discrete definition of individual block pairs seems to be not possible at the moment (see `src/io/discretization/Peridigm_ExodusDiscretization.cpp` lines 744-762)
4. Additional lines might be added by the use of the discretization tools described in section 3.2.

3.1.5 Exemplary input section

Text File – Direct collocation point text file input

XML format

```
<ParameterList name="Discretization">
  <Parameter name="Type" type="string" value="Text File" />
  <Parameter name="Input Mesh File" type="string" value="
    Compression_QS_3x2x2_TextFile.txt"/>
</ParameterList>
```

Free format

-

YAML format

-

Exodus – finite element mesh

XML format

```
<ParameterList name="Discretization">
  <Parameter name="Type" type="string" value="Exodus" />
  <Parameter name="Input Mesh File" type="string" value="Bar.g"/>
  <Parameter name="Omit Bonds Between Blocks" type="string" value="All"
    />
</ParameterList>
```

Free format

```
Discretization
  Type "Exodus"
  Input Mesh File "mesh.g"
  Influence Function "One"
```

YAML format

-

Albany – multiphysics mesh

XML format

-

Free format

-

YAML format

-

PdQuickGrid – Internal mesh generator

XML format

-

Free format

-

YAML format

-

3.1.6 List of examples

Text File – Direct collocation point text file input

- From `test/regression/`:
  - `Compression_QS_3x2x2_TextFile/Compression_QS_3x2x2_TextFile.xml`
- From `test/verification/`:
  - `MultipleHorizons/MultipleHorizons.xml`

Exodus – finite element mesh

- For input parameter *Influence Function*:
  - `/test/verification/Compression_3x1x1_InfluenceFunction/Compression_3x1x1_Influenc`
- For input parameter *Omit Bonds Between Blocks*:
  - `/test/regression/Bar_TwoDisconnectedPieces_QS/Bar.xml`

Albany – multiphysics mesh

PdQuickGrid – Internal mesh generator

3.2. Discretization tools

3.2.1. Preliminaries

Tools to modify the discretization behavior.

3.2.2. Bond Filters

3.2.2.1. Description

A method to omit bonds in the discretization.

3.2.2.2. Code

Dependent on the type of discretization the following classes are used:

- from `src/io/discretization/`:
  - `Peridigm_AlbanyDiscretization.cpp`
  - `Peridigm_AlbanyDiscretization.hpp`
  - `Peridigm_ExodusDiscretization.cpp`
  - `Peridigm_ExodusDiscretization.hpp`
  - `Peridigm_PdQuickGridDiscretization.hpp`
  - `Peridigm_PdQuickGridDiscretization.cpp`
  - `Peridigm_TextFileDiscretization.hpp`
  - `Peridigm_TextFileDiscretization.cpp`

All extend the super-class

- `/src/io/discretization/Peridigm_Discretization.cpp`
- `/src/io/discretization/Peridigm_Discretization.hpp`

Each class uses the method from the super-class :

`PeridigmNS::Discretization::createBondFilters`

3.2.2.3. Input parameters

List

Name	Type	Required	Default	Description
Type	string	✓	-	"Rectangular_Plane | Disk"

For type Rectangular_Plane additionally:

Name	Type	Required	Default	Description
Normal_X	double	✓	-	x-component of the cutting plane normal vector
Normal_Y	double	✓	-	y-component of the cutting plane normal vector
Normal_Z	double	✓	-	z-component of the cutting plane normal vector
Lower_Left_Corner_X	double	✓	-	x component of lower left rectangle corner
Lower_Left_Corner_Y	double	✓	-	y component of lower left rectangle corner
Lower_Left_Corner_Z	double	✓	-	z component of lower left rectangle corner
Bottom_Unit_Vector_X	double	✓	-	x component of rectangle bottom edge direction vector
Bottom_Unit_Vector_Y	double	✓	-	y component of rectangle bottom edge direction vector
Bottom_Unit_Vector_Z	double	✓	-	z component of rectangle bottom edge direction vector
Bottom_Length	double	✓	-	Length of rectangle bottom and top edges
Side_Length	double	✓	-	Length of rectangle left and right edges

For type Disk additionally:

Name	Type	Required	Default	Description
Normal_X	double	✓	-	x-component of the cutting plane normal vector
Normal_Y	double	✓	-	y-component of the cutting plane normal vector
Normal_Z	double	✓	-	z-component of the cutting plane normal vector
Center_X	double	✓	-	x-component of the base point
Center_Y	double	✓	-	y-component of the base point
Center_Z	double	✓	-	z-component of the base point
Radius	double	✓	-	Disk radius

Remarks

1. Bond Filters cause the bonds to not be created in the first place
2. For type
   - "Rectangular_Plane":
     - Define a rectangular cut plane where all bonds are released prior to the simulation.
   - "Disk":
     - Define a circular cut plane where all bonds are released prior to the simulation.
3. It is suggested to define the output variable Number_Of_Neighbors in order to confirm that the bond filter is in the right place. It always seems to take me multiple tries to get it right.

Usage

Rectangular_Plane

The input parameters form a 2D rectangle in 3D space. Bonds that would pass through the rectangle are not formed, which creates a pre-crack.

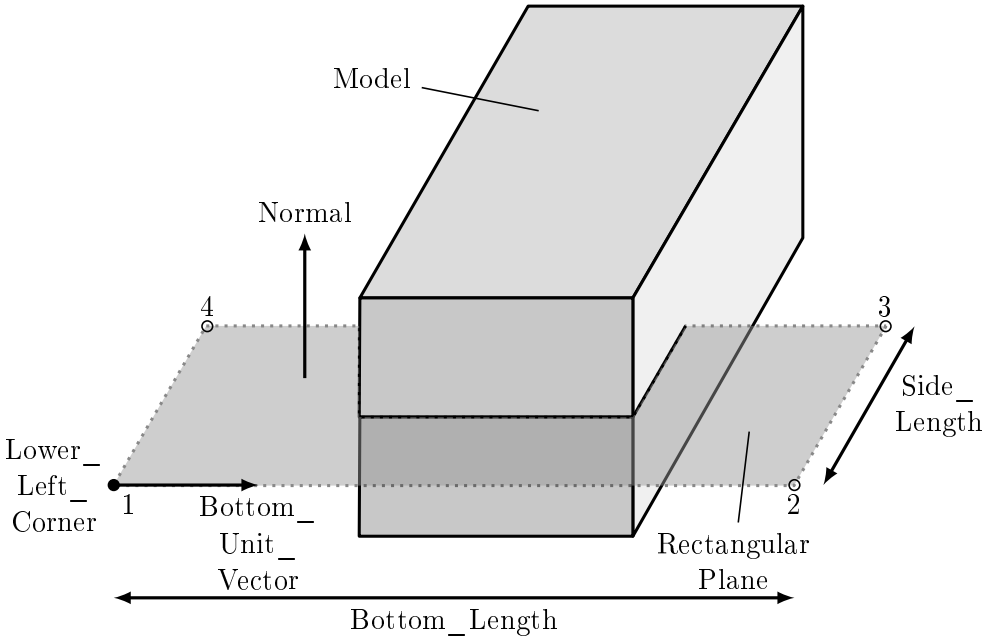


Figure 3.1.: Bond Filter - Rectangular Plane

Lower_ Left_ Corner	Point in 3D which defines the origin of the rectangle in its lower left corner, here named "1".
Bottom_ Unit_ Vector	Vector of unit length in 3D defining the direction along the bottom of the rectangle, here from "1" to "2".
Normal	Vector in 3D that defines the normal direction of the rectangle. From this and *Bottom_ Unit_ Vector* the vector of the other side of the rectangle, here from "1" to "4", is calculated internally using the vector or cross product.
Bottom_ Length	Length of the side along *Bottom_ Unit_ Vector*, so the length between "1" to "2" and "3" to "4".
Side_ Length	Length of the others two sides, here "1" to "4" and "2" to "3".

Disk

The input parameters form a 2D circular disk in 3D space. Bonds that would pass through the rectangle are not formed, which creates a pre-crack.

3.2.2.4. Exemplary input section

XML-format

from `/test/regression/PrecrackedPlate/PrecrackedPlate.xml`:

```
<ParameterList name="Discretization">
  <Parameter name="Type" type="string" value="Exodus" />
  <Parameter name="Input Mesh File" type="string" value="Plate.g"/>
  <ParameterList name="Bond Filters">
    <ParameterList name="My First Bond Filter">
      <Parameter name="Type" type="string" value="Rectangular_Plane"/>
      <Parameter name="Normal_X" type="double" value="0.0"/>
      <Parameter name="Normal_Y" type="double" value="1.0"/>
      <Parameter name="Normal_Z" type="double" value="0.0"/>
      <Parameter name="Lower_Left_Corner_X" type="double" value="-10.0"/>
      <Parameter name="Lower_Left_Corner_Y" type="double" value="0.0"/>
      <Parameter name="Lower_Left_Corner_Z" type="double" value="-10.0"/>
      <Parameter name="Bottom_Unit_Vector_X" type="double" value="1.0"/>
      <Parameter name="Bottom_Unit_Vector_Y" type="double" value="0.0"/>
      <Parameter name="Bottom_Unit_Vector_Z" type="double" value="0.0"/>
      <Parameter name="Bottom_Length" type="double" value="10"/>
      <Parameter name="Side_Length" type="double" value="20"/>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

from `/test/regression/DiskFilter/DiskFilter.xml`:

```
<ParameterList name="Discretization">
  <Parameter name="Type" type="string" value="Exodus" />
  <Parameter name="Input Mesh File" type="string" value="DiskFilter.g"/>
  <ParameterList name="Bond Filters">
    <ParameterList name="My First Bond Filter">
      <Parameter name="Type" type="string" value="Disk"/>
      <Parameter name="Center_X" type="double" value="0.0"/>
      <Parameter name="Center_Y" type="double" value="0.0"/>
      <Parameter name="Center_Z" type="double" value="0.0"/>
      <Parameter name="Normal_X" type="double" value="0.0"/>
      <Parameter name="Normal_Y" type="double" value="1.0"/>
      <Parameter name="Normal_Z" type="double" value="0.0"/>
      <Parameter name="Radius" type="double" value="2.5"/>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Free format

```
Discretization
  ...
  Bond Filters
    bond_filter-1
      Type "Rectangular_Plane"
      Normal_X 0.0
      Normal_Y -1.0
      Normal_Z 0.0
      Lower_Left_Corner_X -0.1
      Lower_Left_Corner_Y 0.0
      Lower_Left_Corner_Z -0.1
      Bottom_Unit_Vector_X 0.0
      Bottom_Unit_Vector_Y 0.0
      Bottom_Unit_Vector_Z 1.0
      Bottom_Length 0.2
      Side_Length 10.1
```

YAML format

-

3.2.2.5. List of examples

- From `test/regression/`:
  - `PrecrackedPlate/PrecrackedPlate.xml`
  - `PrecrackedPlateTwoCracks/PrecrackedPlateTwoCracks.xml`

3.2.3. Influence Function

3.2.3.1. Description

The influence function in the peridynamic theory is used to weight the contribution of all the bonds participating in the computation of volume-dependent properties dependent of the distance to the family origin.

3.2.3.2. Literature

- [13]
- [14]

3.2.3.3. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- from `src/core/`:
  - `Peridigm_Peridigm_InfluenceFunction.cpp`
  - `Peridigm_Peridigm_InfluenceFunction.hpp`

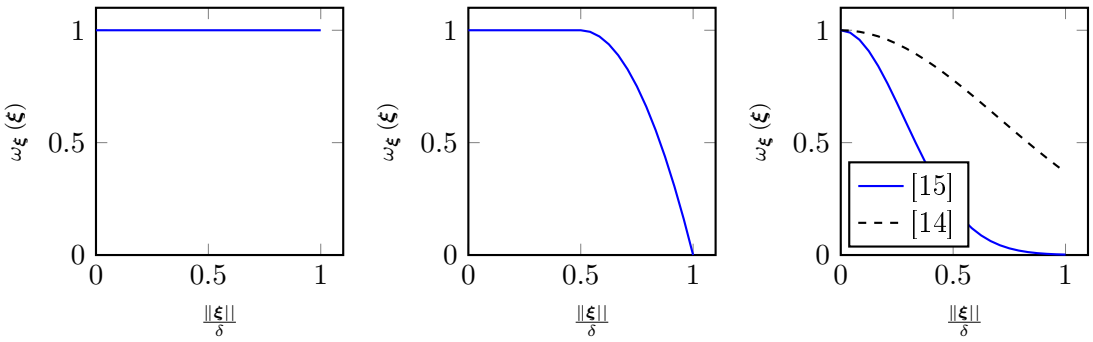
3.2.3.4. Input parameters

List

The influence function is not an individual object. Therefore, no parameter list is required here.

Remarks

1. The influence function is defined for the whole model.
2. Function values are dependent of the bond length $\|\boldsymbol{\xi}\|$ and the local horizon δ
3. The influence functions in Figure 3.2 are pre-implemented. Each unrecognized string is assumed to be an user-defined influence function. The default value is set to *One* in case the keyword is not specified, see `Peridigm.cpp`.



(a) One (b) Parabolic Decay (c) Gaussian

Figure 3.2.: Influence functions

4. For further information on the effect of the influence function see: [13] & [16].
5. Other Influence functions and their usage can be found in [15]. The "Parabolic Decay" is used in [17]. Several different functions can be found in the literature for the "Gaussian" influence function:
   - [14, 18]: $\omega_{\boldsymbol{\xi}}(\boldsymbol{\xi}) = e^{-\left(\frac{\|\boldsymbol{\xi}\|}{\delta}\right)^2}$
   - [15]: $\omega_{\boldsymbol{\xi}}(\boldsymbol{\xi}) = e^{-\left(\frac{\|\boldsymbol{\xi}\|}{0.4\cdot\delta}\right)^2}$

   Here, the version from [15] is used[1].

3.2.3.5. Exemplary input section

XML-format

```
<ParameterList name="Discretization">
  <Parameter name="Type" type="string" value="Exodus" />
  <Parameter name="Input Mesh File" type="string" value="Plate.g"/>
  <Parameter name="Influence Function" type="string" value="Gaussian"/>
</ParameterList>
```

[1] see https://github.com/peridigm/peridigm/issues/42

Free format

-

YAML format

-

3.2.3.6. List of examples

-

3.3. Material models

3.3.1. Preliminaries

Generally, material models or more generelly peridynamic constitutive laws are divided by the underlying peridynamic formulation and can be grouped into two categories:

Bond-based bond forces depend only on a single pair of material points
State-based bond forces depend on deformations of all neighboring material points

A rough classification of material models in Peridigm can be found in Figure 3.3.

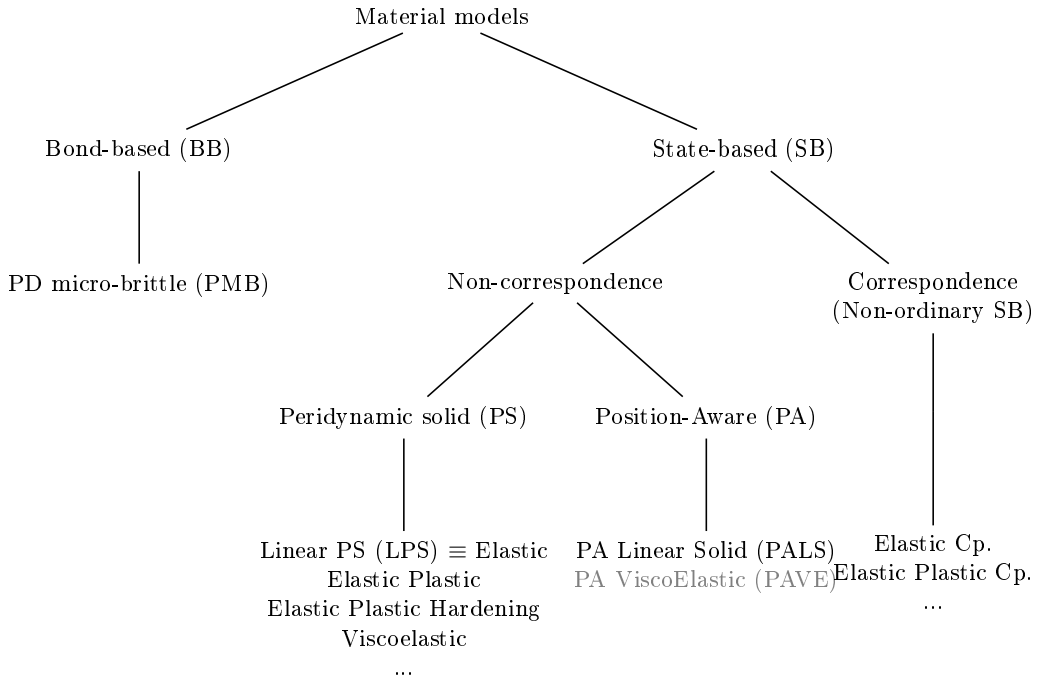


Figure 3.3.: Classification of peridynamic material models

Bond-based material models are not discussed here any further. Peridynamic state-based material models are further divided in two main classes, the

- non-correspondence and
- correspondence

materials. The non-correspondence is material created directly in the peridynamic formulation. The correspondence material is formulated in the classical continuum mechanical way. Both material classes will/should evaluate the same result for the same material modeled if surface effects have no influence. However, the correspondence material gives

additional outputs as Cauchy stresses. This makes the results more comparable to the standard formulation.

3.3.1.1. Comments on non-correspondence material models

Description

There are sub-classes of non-correspondence material models. The first material model implementation is the peridynamic micro-brittle (PMB) material for bond-based peridynamics. Due to the limitations of bond-based peridynamics, this is not further discussed here.

State-based non-correspondence material models are the linear peridynamic solid (LPS) model and its extension to other types of material behaviour, like plasticity or viscoelasticity. These materials suffer from one large disadvantage, the underlying mathematical models assume that all points within are in the bulk. However, points near the surface are missing bonds. Missing bonds imply and induce incorrect material properties. Thus, these models are consistent.

To overcome this limitation, the so-called *Position-Aware* class of material models was developed.

Remarks

- If not stated otherwise, peridynamic solid non-correspondence (PS) materials in Peridigm do not have surface correction implemented.
- Position-aware (PA) materials were developed to account for missing bonds on surfaces removing the need for auxiliary surface correction techniques.

3.3.1.2. Comments on correspondence material models

- Do not require surface correction
- Might suffer from zero-energy modes
- Each family needs at least 3 non-collinear bonds active, otherwise the deformation gradient can not be calculated and the solution fails.
- Therefore, the bond check algorithm included in the interface aware damage model (cf. section 3.4.4) has to be included if a damage model should be used.

3.3.2. Elastic

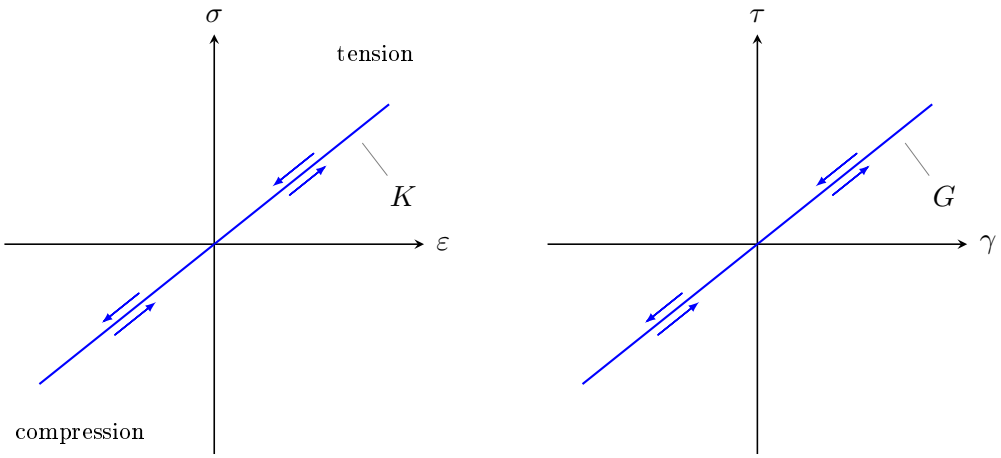
3.3.2.1. Description

An isotropic, linear elastic material model. This material is also commonly known as the linear peridynamic solid (LPS) material model. The model does not support a flexible horizon, therefore a constant horizon for each block is assumed.

3.3.2.2. Literature

↗ [19]

3.3.2.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.4.: Linear-elastic material model

3.3.2.4. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_ElasticMaterial.cpp`
  - `/src/materials/Peridigm_ElasticMaterial.hpp`
- Computation:
  - `/src/materials/elastic.cxx`
  - `/src/materials/elastic.h`

3.3.2.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Elastic"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction
Thermal Expansion Coefficient	double	-	-	Thermal Expansion Coefficient
Apply Shear Correction Factor	bool	-	true	
Plane Strain[4]	bool	-	false	Plane strain formulation
Plane Stress[4]	bool	-	false	Plane stress formulation

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)}$$

Internally, the engineering constants are converted to (K,G).

3. Consider the general remarks on non-correspondence materials in section 3.3.1.1
4. Activate plane strain or plane stress formulation for 2D analyses, cf. Equation (8) in [20]. Both or mutually exclusive. If none of the two parameters is used, the default 3D formulation is applied.

3.3.2.6. Exemplary input section

XML-format

Using engineering constants:

```
<ParameterList name="Materials">
  <ParameterList name="My Material">
    <Parameter name="Material Model" type="string" value="Elastic"/>
    <Parameter name="Density" type="double" value="8.0e-9"/>
    <Parameter name="Young's Modulus" type="double" value="192e3"/>
    <Parameter name="Poisson's Ratio" type="double" value="0.33333"/>
    <Parameter name="Plane Strain" type="bool" value="true"/>
    <Parameter name="Plane Stress" type="bool" value="false"/>
  </ParameterList>
</ParameterList>
```

Using volumetric and shear stiffness: from Link

```
<ParameterList name="Materials">
  <ParameterList name="My Elastic Material">
    <Parameter name="Material Model" type="string" value="Elastic"/>
    <Parameter name="Apply Shear Correction Factor" type="bool" value="
   false"/>
    <Parameter name="Density" type="double" value="2200.0"/>
    <Parameter name="Bulk Modulus" type="double" value="14.90e9"/>
    <Parameter name="Shear Modulus" type="double" value="8.94e9"/>
     <Parameter name="Plane Strain" type="bool" value="true"/>
    <Parameter name="Plane Stress" type="bool" value="false"/>
  </ParameterList>
</ParameterList>
```

```
<ParameterList name="Materials">
  <ParameterList name="My Elastic Material">
    <Parameter name="Material Model" type="string" value="Elastic"/>
    <Parameter name="Density" type="double" value="7800.0"/>
    <Parameter name="Bulk Modulus" type="double" value="130.0e9"/>
    <Parameter name="Shear Modulus" type="double" value="78.0e9"/>
```

```
    <Parameter name="Thermal Expansion Coefficient" type="double" value
   ="10.0e-6"/>
    <Parameter name="Plane Strain" type="bool" value="true"/>
    <Parameter name="Plane Stress" type="bool" value="false"/>
  </ParameterList>
</ParameterList>
```

from `/test/examples/fragmenting_cylinder/fragmenting_cylinder.peridigm`

Free format

-

YAML format

-

3.3.2.7. Possible output variables for the material model

- Bond_Damage
- Coordinates
- Damage
- Deviatoric_Plastic_Extension
- Dilatation
- Force_Density
- Lambda
- Model_Coordinates
- Partial_Stress
- Surface_Correction_Factor
- Temperature_Change
- Volume
- Weighted_Volume

3.3.2.8. List of examples

- From `test/regression/`:
  - `Bar_OneBlock_OneMaterial_QS/Bar.xml`
  - `Body_Force/Body_Force_Explicit.xml`

3.3.3. Elastic Bond Based

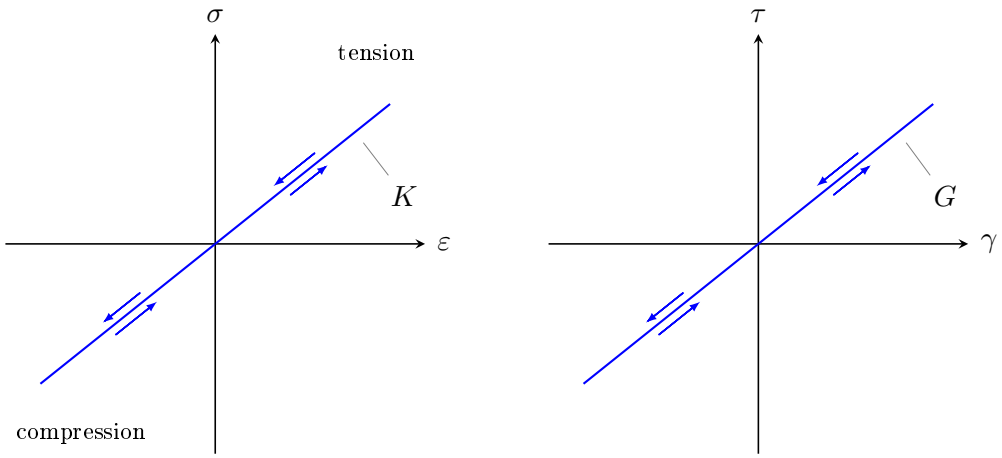
3.3.3.1. Description

An isotropic, linear elastic bond-based material model.

3.3.3.2. Literature

↗ [21]

3.3.3.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.5.: Linear-elastic material model

3.3.3.4. Code

Release version

Available from version 1.5.

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_ElasticBondBasedMaterial.cpp`
  - `/src/materials/Peridigm_ElasticBondBasedMaterial.hpp`
- Computation:
  - `/src/materials/elastic_bond_based.cxx`
  - `/src/materials/elastic_bond_based.h`

3.3.3.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Elastic Bond Based"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2,3]	-	Volumetric elasticity

Remarks

1. Only the bulk modulus, K, can be used as elastic constant.
2. The Poisson's Ratio is set to a fixed value of $\frac{1}{4}$. The validity range of this value is discussed in Table 2.3
3. The *Shear Modulus* is calculated from

$$G = \frac{3K\,(1-2\nu)}{2\cdot(1+\nu)}$$

4. Shear Correction Factor is not supported for the material model
5. Thermal expansion is not currently supported for the material model

3.3.3.6. Exemplary input section

XML-format

-

Free format

-

YAML format

-

3.3.3.7. Possible output variables for the material model

3.3.3.8. List of examples

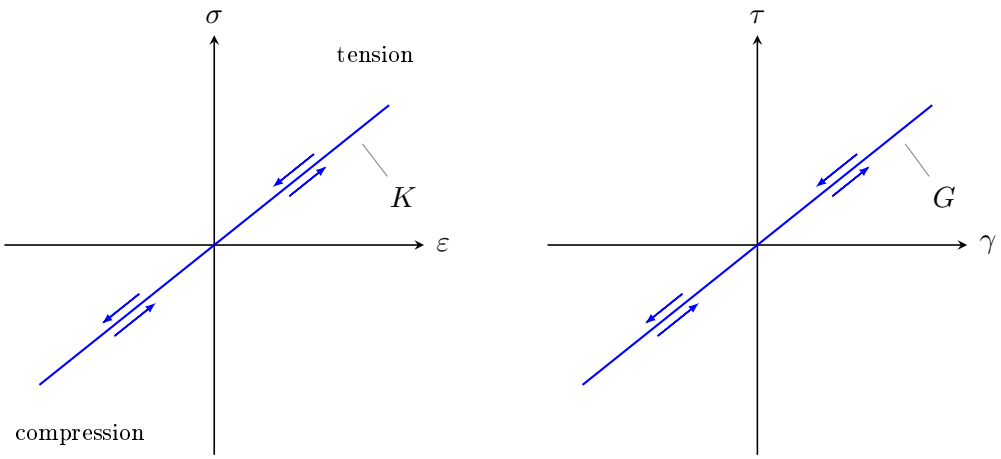
-

3.3.4. Elastic Correspondence

3.3.4.1. Description

An isotropic, linear elastic correspondence material model.

3.3.4.2. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.6.: Linear-elastic correspondence material model

3.3.4.3. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_ElasticCorrespondenceMaterial.cpp`
  - `/src/materials/Peridigm_ElasticCorrespondenceMaterial.hpp`

- Computation:
  - /src/materials/elastic_correspondence.cxx
  - /src/materials/elastic_correspondence.h

3.3.4.4. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Elastic Correspondence"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction
Hourglass Coefficient[3]	double	✓	-	
Compute Partial Stress	bool	-	?	

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

   $$G = \frac{E}{2 \cdot (1 + \nu)}$$

   Internally, the engineering constants are converted to (K,G).
3. Hourglass coefficient is usually between 0.0 and 0.05
4. Automatic Differentiation is not supported for the Correspondence material model
5. Shear Correction Factor is not supported for the Correspondence material model
6. Thermal expansion is not currently supported for the Correspondence material model
7. Consider the general remarks on correspondence materials in section 3.3.1.2

3.3.4.5. Exemplary input section

XML-format

-

Free format

```
Materials
  My Material
    Material Model "Elastic Correspondence"
    Density 8.0
    Bulk Modulus 1.500e12
    Shear Modulus 6.923e11
    Hourglass Coefficient 0.02
```

YAML format

-

3.3.4.6. Possible output variables for the material model

- Bond_Damage
- Coordinates
- Damage
- Deviatoric_Plastic_Extension
- Dilatation
- Force_Density
- Lambda
- Model_Coordinates
- Partial_Stress
- Surface_Correction_Factor
- Temperature_Change
- Volume
- Weighted_Volume

Additional correspondence material output variables:

- Unrotated_Cauchy_Stress
- Unrotated_Rate_Of_Deformation

3.3.4.7. List of examples

- From `examples/`:
  - `tensile_test/tensile_test.peridigm`

3.3.5. Elastic Partial Volume

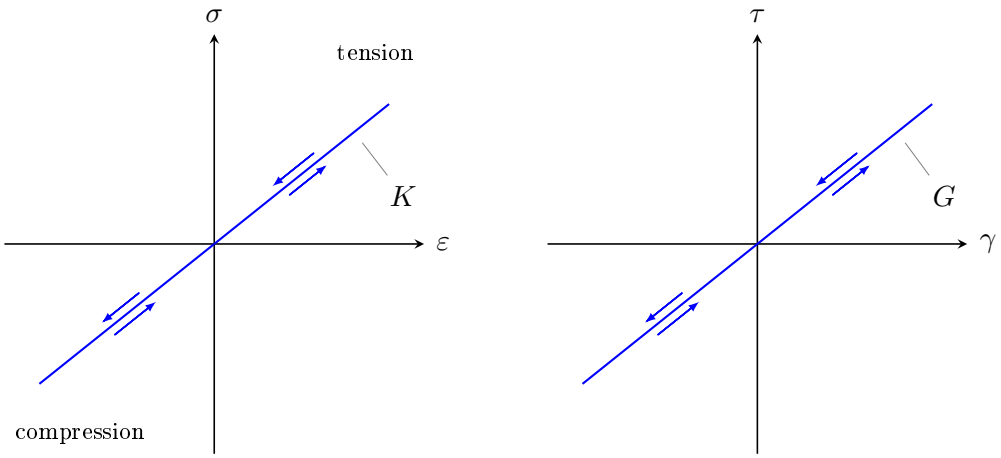
3.3.5.1. Description

An isotropic, partial volume linear elastic material model.

3.3.5.2. Literature

-

3.3.5.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.7.: Partial-volume linear-elastic material model

3.3.5.4. Code

Release version

?

Required compiler options

use `-D USE_PV:BOOL=ON`

Routines

3.3.5.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Elastic Partial Volume"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)}$$

   Internally, the engineering constants are converted to (K,G).
3. Consider the general remarks on non-correspondence materials in section 3.3.1.1

3.3.5.6. Exemplary input section

XML-format

-

Free format

```
Materials
  LPS Material
    Material Model "Elastic Partial Volume"
    Density 8.05
    Bulk Modulus 160.0e10
    Shear Modulus 79.3e10
```

YAML format

-

3.3.5.7. Additional output variables for the material model

3.3.5.8. List of examples

- From `test/verification/`:
  - `LinearLPSBar/LinearLPSBar.peridigm`

3.3.6. Pals - Position Aware Linear Solid

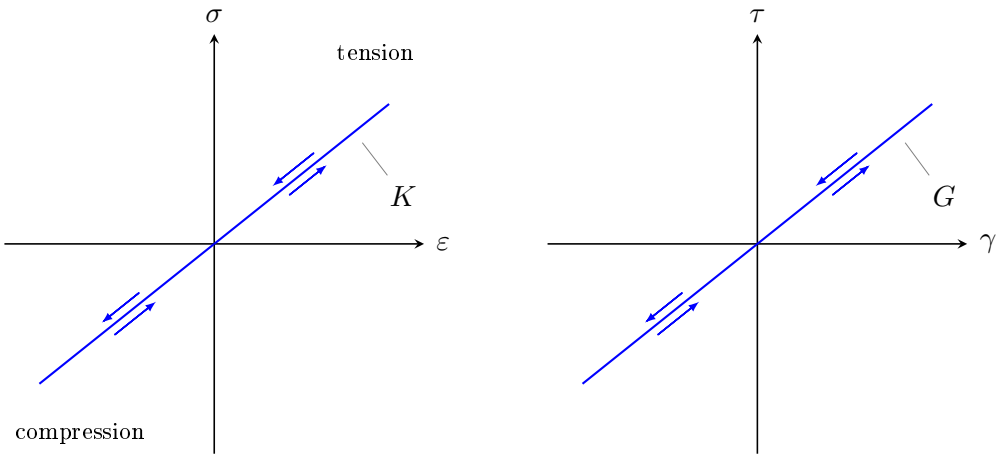
3.3.6.1. Description

An isotropic, position-aware linear elastic material model.

3.3.6.2. Literature

↗ [22]

3.3.6.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.8.: Position-aware linear-elastic material model

3.3.6.4. Code

Release version

Available from version 1.5.

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_Pals_Model.cpp`
  - `/src/materials/Peridigm_Pals_Model.hpp`
- Computation:
  - `/src/materials/pals.cxx`
  - `/src/materials/pals.h`

3.3.6.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Pals"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction
Dilatation Influence Function	string[4]	-	"One"	"One" | "Parabolic Decay" | "Gaussian" | User defined
Deviatoric Influence Functions	string[4]	-	"One"	"One" | "Parabolic Decay" | "Gaussian" | User defined

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)}$$

Internally, the engineering constants are converted to (K,G).

3. Consider the general remarks on non-correspondence materials in section 3.3.1.1
4. See section 3.2.3 & `/src/core/Peridigm_InfluenceFunction.cpp/hpp`. Each unrecognized string is assumed to be an user-defined influence function.

3.3.6.6. Exemplary input section

XML-format

```
<ParameterList name="Materials">
  <ParameterList name="My Material">
    <Parameter name="Material Model" type="string" value="Pals"/>
    <Parameter name="Dilatation Influence Function" type="string" value="
    One"/>
    <Parameter name="Deviatoric Influence Function" type="string" value="
    One"/>
    <Parameter name="Density" type="double" value="2700.0"/>
    <Parameter name="Bulk Modulus" type="double" value="67.55e9"/>
    <Parameter name="Shear Modulus" type="double" value="25.90e9"/>
  </ParameterList>
</ParameterList>
```

Free format

```
Materials
  testadhesivematerial
    Material Model "Pals"
    Density {MAT_DENSITY}
    Bulk Modulus 3202.81124497992
    Shear Modulus 1195.6521739130435
    Dilatation Influence Function "One"
    Deviatoric Influence Function "One"
```

YAML format

-

3.3.6.7. Additional output variables for the material model

- Deviatoric_Normalization
- Dilatation
- Dilatation_Normalization
- Lagrange_Multiplier_Dilatation_1-6

3.3.6.8. List of examples

- From `test/regression/`:
  - `Pals_Simple_Shear/Pals_Simple_Shear.xml`

3.3.7. Linear LPS Partial Volume

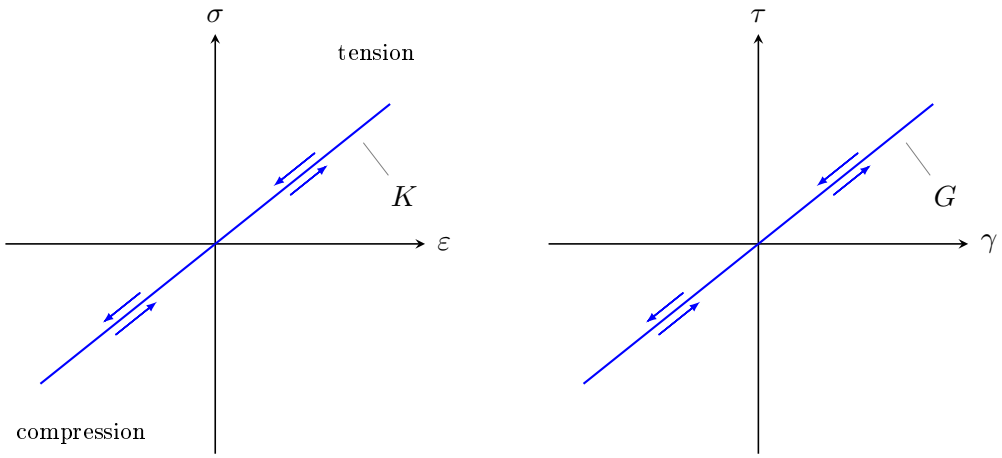
3.3.7.1. Description

-

3.3.7.2. Literature

↗ -

3.3.7.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.9.: Partial-volume linear-peridynamic solid/elastic material model

3.3.7.4. Code

Release version

Available from version 1.4.

Required compiler options

use `-D USE_PV:BOOL=ON`

Routines

- IO:
  - `/src/materials/Peridigm_LinearLPSPVMaterial.cpp`
  - `/src/materials/Peridigm_LinearLPSPVMaterial.hpp`
- Computation:
  - `/src/materials/linear_lps_pv.cxx`
  - `/src/materials/linear_lps_pv.h`

3.3.7.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Linear LPS Partial Volume"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)}$$

   Internally, the engineering constants are converted to (K,G).
3. Consider the general remarks on non-correspondence materials in section 3.3.1.1

3.3.7.6. Exemplary input section

XML-format

-

Free format

```
Materials
  Linearized LPS Material
    Material Model "Linear LPS Partial Volume"
    Density 8.05
    Bulk Modulus 160.0e10
    Shear Modulus 79.3e10
```

YAML format

-

3.3.7.7. Possible output variables for the material model

3.3.7.8. List of examples

- From `test/verification/`:
  - `LinearLPSBar/LinearLPSBar.peridigm`

3.3.8. LCM

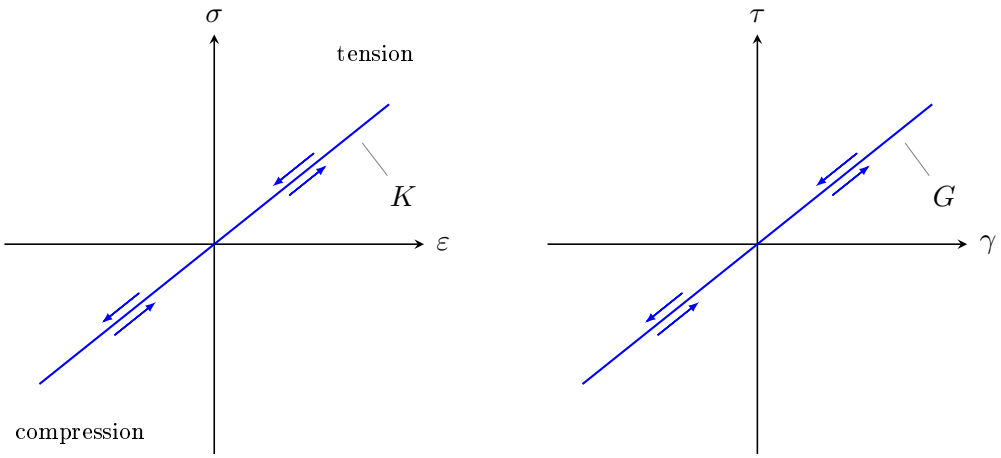
3.3.8.1. Description

An isotropic, linear elastic material model. Purpose of this material model is a non-ordinary state-based interface to classical material models in the Laboratory for Computational Mechanics (LCM) (Albany).

3.3.8.2. Literature

↗ -

3.3.8.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.10.: Linear-elastic material model

3.3.8.4. Code

Release version

Available from version 1.2.

Required compiler options

use `-D USE_LCM:BOOL=ON` and specify/adapt the following path:

```
-D USE_LCM:BOOL=ON
-D LCM_INCLUDE_DIR:PATH=/Users/djlittl/Albany/src/LCM
-D LCM_LIBRARY_DIR:PATH=/Users/djlittl/Albany/GCC_4.7.2_OPT/src
```

Routines

- IO:
  - `/src/materials/Peridigm_LCMMaterial.cpp`
  - `/src/materials/Peridigm_LCMMaterial.hpp`
- Computation:
  - ?

3.3.8.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	-	-	Material type "LCM"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1]	-	Volumetric elasticity
Shear Modulus	double	✓[1]	-	Shear elasticity or engineering constant for shear stiffness

Remarks

1. The stiffness can be only defined by the Lamé coefficients (K,G)
2. Thermal expansion is not currently supported for the material model

3.3.8.6. Exemplary input section

XML-format

-

Free format

-

YAML format

-

3.3.8.7. Possible output variables for the material model

3.3.8.8. List of examples

-

3.3.9. Elastic Plastic

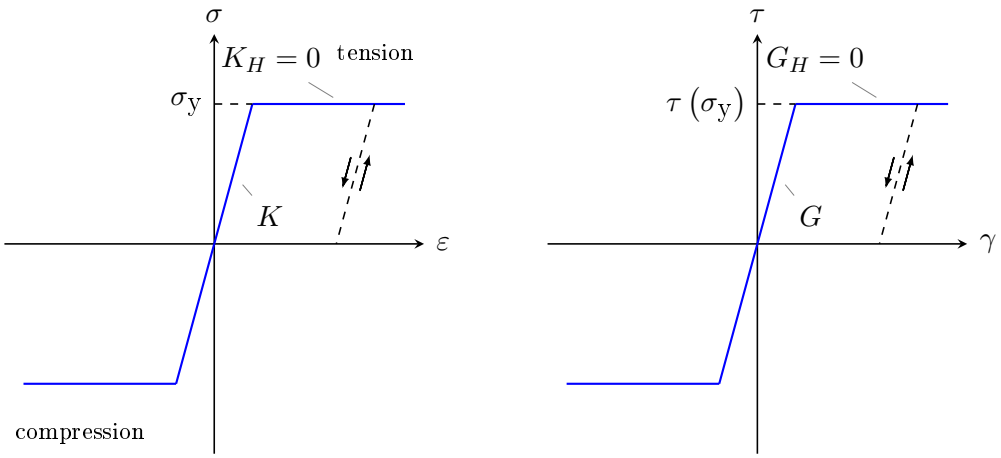
3.3.9.1. Description

An isotropic, linear elastic perfectly plastic material model.

3.3.9.2. Literature

↗ [23]

3.3.9.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.11.: Linear-elastic perfectly-plastic material model

3.3.9.4. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- IO:
  - /src/materials/Peridigm_ElasticPlasticMaterial.cpp
  - /src/materials/Peridigm_ElasticPlasticMaterial.hpp
- Computation:
  - /src/materials/elastic_plastic.cxx
  - /src/materials/elastic_plastic.h

3.3.9.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Elastic Plastic"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction
Yield Stress	double	✓	-	Yield stress
Apply Shear Correction Factor	bool	-	true	
Disable Plasticity	bool	-	false	

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)}$$

Internally, the engineering constants are converted to (K,G).

3. Yield stress is an equivalent uniaxial stress and therefore applicable to axial, shear as well as mixed stress states.
4. Thermal expansion is not currently supported for the Elastic Plastic material model
5. Consider the general remarks on non-correspondence materials in subsubsection 3.3.1.1

3.3.9.6. Exemplary input section

XML-format

From `ep_cube.xml`:

```
<ParameterList name="Materials">
  <ParameterList name="Elastic Plastic">
    <Parameter name="Density" type="double" value="2.7e-3"/>
    <Parameter name="Bulk Modulus" type="double" value="67549.0"/>
    <Parameter name="Shear Modulus" type="double" value="25902.2"/>
    <Parameter name="Yield Stress" type="double" value="351.79"/>
    <Parameter name="Apply Shear Correction Factor" type="bool" value="
    false"/>
  </ParameterList>
</ParameterList>
```

```
<ParameterList name="RightMaterial">
  <Parameter name="Material Model" type="string" value="Elastic Plastic
    "/>
  <Parameter name="Density" type="double" value="8.0e-9"/>
  <Parameter name="Bulk Modulus" type="double" value="1.515e5"/>
  <Parameter name="Shear Modulus" type="double" value="7.813e4"/>
  <Parameter name="Yield Stress" type="double" value="1.0e5"/>
  <Parameter name="Disable Plasticity" type="bool" value="true"/>
  <Parameter name="Apply Shear Correction Factor" type="bool" value="
    false"/>
</ParameterList>
```

Free format

-

YAML format

-

3.3.9.7. Possible output variables for the material model

- Bond_Damage
- Coordinates
- Damage
- Deviatoric_Plastic_Extension
- Dilatation
- Force_Density
- Lambda
- Model_Coordinates
- Volume
- Weighted_Volume

3.3.9.8. List of examples

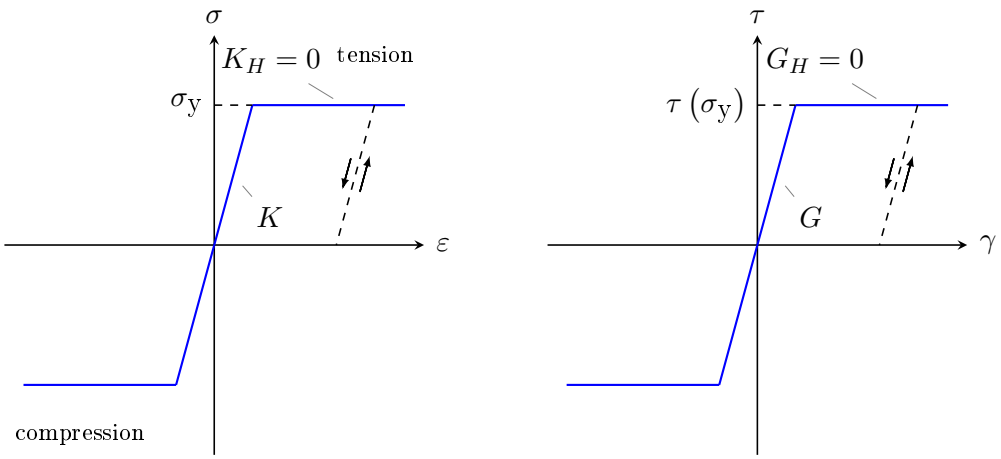
- From `test/regression/`:
  - `Bar_TwoBlocks_TwoDifferentMaterial_QS/Bar.xml`
- From `test/verification/`:
  - `ep_cube/ep_cube.xml`

3.3.10. Elastic Plastic Correspondence

3.3.10.1. Description

An isotropic, linear elastic perfectly plastic correspondence material model.

3.3.10.2. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.12.: Linear-elastic perfectly-plastic correspondence material model

3.3.10.3. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_ElasticPlasticCorrespondenceMaterial.cpp`
  - `/src/materials/Peridigm_ElasticPlasticCorrespondenceMaterial.hpp`

- Computation:
  - `/src/materials/elastic_plastic_correspondence.cxx`
  - `/src/materials/elastic_plastic_correspondence.h`

3.3.10.4. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	-	-	Material type "Elastic Plastic Correspondence"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction
Yield Stress	double	✓	-	Yield stress
Apply Shear Correction Factor	bool	-	true	
Hourglass Coefficient[4]	double	✓		
Disable Plasticity	bool	-	false	

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)}$$

   Internally, the engineering constants are converted to (K,G).
3. Yield stress is an equivalent uniaxial stress and therefore applicable to axial, shear as well as mixed stress states.
4. Hourglass coefficient is usually between 0.0 and 0.05

5. Automatic Differentiation is not supported for the correspondence material model
6. Shear Correction Factor is not supported for the correspondence material model
7. Thermal expansion is not currently supported for the correspondence material model
8. Consider the general remarks on correspondence materials in section 3.3.1.2

3.3.10.5. Exemplary input section

XML-format

```
<ParameterList name="Materials">
  <ParameterList name="My Elastic Plastic Correspondence Material">
    <Parameter name="Material Model" type="string" value="Elastic Plastic
     Correspondence"/>
    <Parameter name="Density" type="double" value="7800.0"/>
    <Parameter name="Young's Modulus" type="double" value="200.0e9"/>
    <Parameter name="Poisson's Ratio" type="double" value="0.0"/>   <!--
   One-dimensional simulation -->
    <Parameter name="Hourglass Coefficient" type="double" value="0.0"/>
    <Parameter name="Yield Stress" type="double" value="150.0e6"/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

3.3.10.6. Possible output variables for the material model

- Bond_Damage
- Coordinates
- Damage
- Deviatoric_Plastic_Extension
- Dilatation
- Force_Density
- Lambda
- Model_Coordinates
- Volume
- Weighted_Volume

Additional correspondence material output variables:

- Equivalent_Plastic_Strain
- Unrotated_Cauchy_Stress
- Unrotated_Rate_Of_Deformation
- Von_Mises_Stress

3.3.10.7. List of examples

- From `test/verification/`:
  - `ElasticPlasticCorrespondenceFullyPrescribedTension/ElasticPlasticCorrespondenceFu`

3.3.11. Pressure Dependent Elastic Plastic

3.3.11.1. Description

3.3.11.2. Literature

↗ [24]

3.3.11.3. Stiffness model sketch

-

3.3.11.4. Code

Release version

Available from version 1.2.

Required compiler options

use `-D USE_CJL:BOOL=ON`

Routines

3.3.11.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	-	-	Material type "Pressure Dependent Elastic Plastic"

Remarks

3.3.11.6. Exemplary input section

XML-format

-

Free format

-

YAML format

-

3.3.11.7. Possible output variables for the material model

3.3.11.8. List of examples

- -

3.3.12. Elastic Plastic Hardening

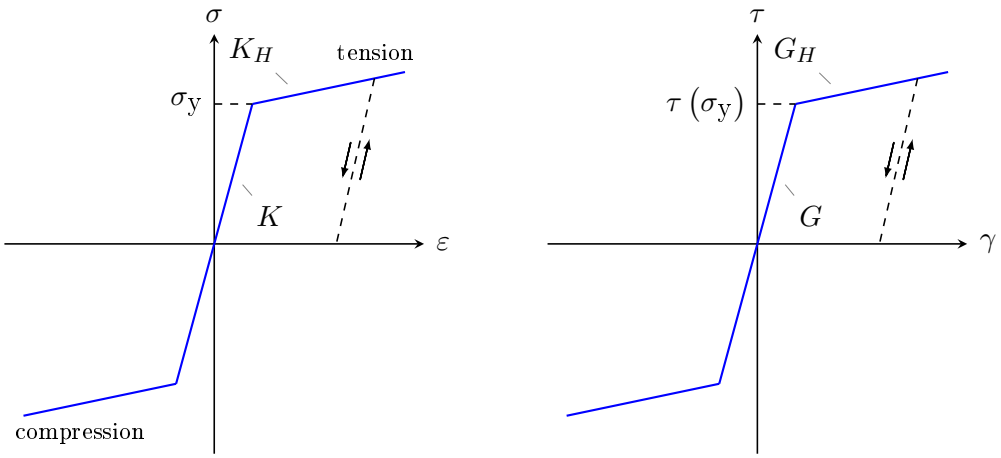
3.3.12.1. Description

An isotropic, linear-elastic linear-hardening material model.

3.3.12.2. Literature

↗ [25]

3.3.12.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.13.: Linear-elastic linear-hardening material model

3.3.12.4. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_ElasticPlasticHardeningMaterial.cpp`
  - `/src/materials/Peridigm_ElasticPlasticHardeningMaterial.hpp`
- Computation:
  - `/src/materials/elastic_plastic_hardening.cxx`
  - `/src/materials/elastic_plastic_hardening.h`

3.3.12.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	-	-	Material type "Elastic Plastic Hardening"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's Modulus	double	✓[1,2]	-	Engineering constant for axial stiffness
Poisson's Ratio	double	✓[1,2]	-	Engineering constant for transverse contraction
Yield Stress	double	✓	-	Yield stress
Hardening modulus	double	✓	-	Linear hardening modulus
Apply Shear Correction Factor	bool	-	true	
Disable Plasticity	bool	-	false	

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)} \qquad G_H = \frac{E_H}{2 \cdot (1 + \nu)}$$

Internally, the engineering constants are converted to (K,G).
3. Yield stress is an equivalent uniaxial stress and therefore applicable to axial, shear as well as mixed stress states.
4. Thermal expansion is not currently supported for the Elastic Plastic Hardening material model
5. Consider the general remarks on non-correspondence materials in subsubsection 3.3.1.1

3.3.12.6. Exemplary input section

XML-format

Free format

-

YAML format

-

3.3.12.7. Possible output variables for the material model

- Bond_Damage
- Coordinates
- Damage
- Deviatoric_Plastic_Extension
- Dilatation
- Force_Density
- Lambda
- Model_Coordinates
- Surface_Correction_Factor
- Volume
- Weighted_Volume

3.3.12.8. List of examples

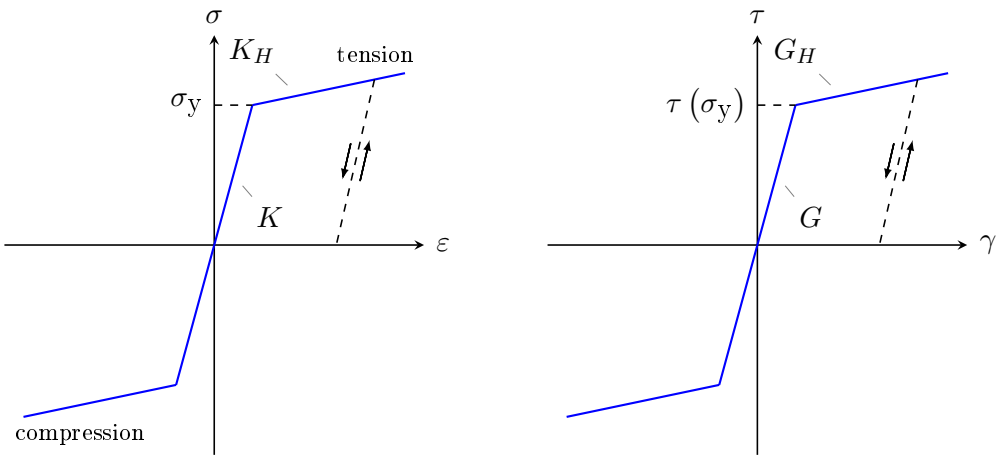
-

3.3.13. Elastic Plastic Hardening Correspondence

3.3.13.1. Description

A linear-elastic linear-hardening correspondence material model.

3.3.13.2. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.14.: Linear-elastic linear-hardening correspondence material model

3.3.13.3. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- IO:
  - /src/materials/Peridigm_IsotropicHardeningPlasticCorrespondenceMaterial.cpp
  - /src/materials/Peridigm_IsotropicHardeningPlasticCorrespondenceMaterial.hpp

- Computation:
  - /src/materials/isotropic_hardening_correspondence.cxx
  - /src/materials/isotropic_hardening_correspondence.h

3.3.13.4. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	-	-	Material type "Isotropic Hardening Correspondence"
Density	double	✓	-	Material density
Bulk modulus	double	✓[1,2]	-	Volumetric elasticity
Shear Modulus	double	✓[1,2]	-	Shear elasticity or engineering constant for shear stiffness
Young's modulus	double	✓[1,2]	-	Axial stiffness
Poisson's ratio	double	✓[1,2]	-	Transverse contraction
Yield Stress	double	✓	-	Yield stress
Hardening modulus	double	✓	-	Linear hardening modulus
Apply Shear Correction Factor	bool	-	true	
Hourglass Coefficient[4]	double	✓		
Disable Plasticity	bool	-	false	

Remarks

1. The stiffness can be defined by either elastic modulus combination: Volumetric and shear elasticity (K,G) or the engineering constants (E,ν,G)
2. In case engineering constants are used, only two of the three values *Young's Modulus*, *Poisson's Ratio* and *Shear Modulus* have to be specified. The missing value is calculated from

$$G = \frac{E}{2 \cdot (1 + \nu)} \qquad G_H = \frac{E_H}{2 \cdot (1 + \nu)}$$

   Internally, the engineering constants are converted to (K,G).
3. Yield stress is an equivalent uniaxial stress and therefore applicable to axial, shear as well as mixed stress states.
4. Hourglass coefficient is usually between 0.0 and 0.05
5. Automatic Differentiation is not supported for the correspondence material model

6. Shear Correction Factor is not supported for the correspondence material model
7. Thermal expansion is not currently supported for the correspondence material model
8. Consider the general remarks on correspondence materials in section 3.3.1.2

3.3.13.5. Exemplary input section

XML-format

```
<ParameterList name="Materials">
  <ParameterList name="My Elastic Plastic Hardening Correspondence
   Material">
    <Parameter name="Material Model" type="string" value="Isotropic
   Hardening Correspondence"/>
    <Parameter name="Density" type="double" value="7800.0"/>
    <Parameter name="Young's Modulus" type="double" value="211.0e9"/>
    <Parameter name="Poisson's Ratio" type="double" value="0.0"/>   <!--
   One-dimensional simulation -->
    <Parameter name="Yield Stress" type="double" value="460.0e6"/>
    <Parameter name="Hardening Modulus" type="double" value="500.0e7"/>
    <Parameter name="Hourglass Coefficient" type="double" value="0.0"/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

3.3.13.6. Possible output variables for the material model

- Bond_Damage
- Coordinates
- Damage
- Deviatoric_Plastic_Extension
- Dilatation
- Force_Density
- Lambda
- Model_Coordinates
- Surface_Correction_Factor
- Volume
- Weighted_Volume

Additional correspondence material output variables:

- Equivalent_Plastic_Strain
- Unrotated_Cauchy_Stress
- Unrotated_Rate_Of_Deformation
- Von_Mises_Stress

3.3.13.7. List of examples

- From `test/verification/`:
  - `IsotropicHardeningPlasticFullyPrescribedTension_NoFlaw/IsotropicHardeningPlasticF NoFlaw.xml`
  - `IsotropicHardeningPlasticFullyPrescribedTension_WithFlaw/IsotropicHardeningPlasti WithFlaw.xml`

3.3.14. Viscoelastic

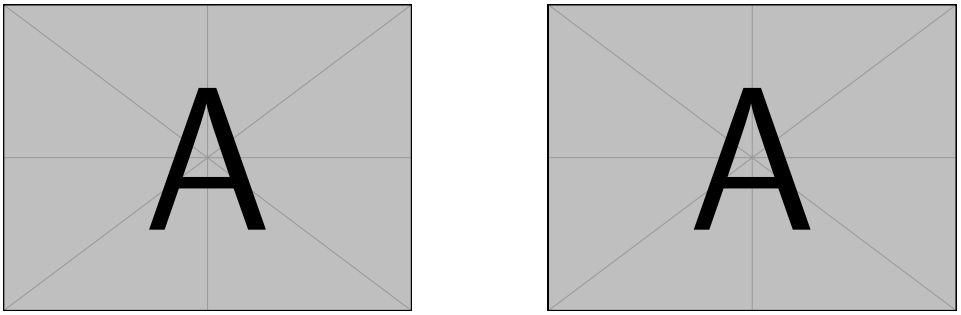
3.3.14.1. Description

An isotropic, viscoelastic material model.

3.3.14.2. Literature

- [26]
- [2]

3.3.14.3. Stiffness model sketch



(a) Axial (b) Shear

Figure 3.15.: Viscoelastic material model

3.3.14.4. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_ViscoelasticMaterial.cpp`
  - `/src/materials/Peridigm_ViscoelasticMaterial.hpp`
- Computation:
  - `/src/materials/viscoelastic.cxx`
  - `/src/materials/viscoelastic.h`

3.3.14.5. Input parameters

List

Name	Type	Required	Default	Description
Material Model	string	✓	-	Material type "Viscoelastic"
Density	double	✓	-	Material density
lambda_i	double	✓[1]	-	Rate of relaxation
tau b	double	✓	-	Relaxation time constant

Remarks

1. $0.0 \leq \lambda_i \leq 1.0$
2. Automatic Differentiation is not supported for the Viscoelastic material model
3. Shear Correction Factor is not supported for the Viscoelastic material model
4. Thermal expansion is not currently supported for the Viscoelastic material model

3.3.14.6. Exemplary input section

XML-format

-

Free format

-

YAML format

-

3.3.14.7. Possible output variables for the material model

- Bond_Damage
- Coordinates
- Damage
- Deviatoric_Back_Extension
- Dilatation
- Force_Density
- Model_Coordinates
- Volume
- Weighted_Volume

3.3.14.8. List of examples

3.3.15. Viscoplastic Needleman Correspondence

3.3.15.1. Description

-

3.3.15.2. Literature

- [27]?
- Ted Belytschko, Huai-Yang Chiang, Edward Plaskacz: High resolution two-dimensional shear band computations: imperfections and mesh dependence, 1994

3.3.15.3. Stiffness model sketch

-

3.3.15.4. Code

Release version

-

Required compiler options

-

Routines

- IO:
  - `/src/materials/Peridigm_ViscoPlasticNeedlemanCorrespondenceMaterial.cpp`
  - `/src/materials/Peridigm_ViscoPlasticNeedlemanCorrespondenceMaterial.hpp`
- Computation:
  - `/src/materials/viscoplastic_needleman_correspondence.cxx`
  - `/src/materials/viscoplastic_needleman_correspondence.h`

3.3.15.5. Input parameters

List

-

Remarks

-

3.3.15.6. Exemplary input section

XML-format

-

Free format

-

YAML format

-

3.3.15.7. Possible output variables for the material model

-

3.3.15.8. List of examples

- From `test/verification/`:
  - `ViscoplasticNeedlemanFullyPrescribedTension_NoFlaw/ViscoplasticNeedlemanFullyPres`
    `NoFlaw.xml ViscoplasticNeedlemanFullyPrescribedTension_WithFlaw/ViscoplasticNeedl`
    `WithFlaw.xml`

3.3.16. Vector Poisson

3.3.17. Multiphysics Elastic

3.4. Damage Models

3.4.1. Preliminaries

The damage models determine whether a bond is damaged or not. Failure of a bond is irreversible. Currently, degradation or material softening of a bond is not implemented. However, successive failure of individual bonds leads to an degradation for the integral material response.

3.4.2. Critical Stretch

3.4.2.1. Description

The critical stretch criterion compares the tensile stretch of a local bond with a critical value. After reaching the critical stretch there is no softening or stiffness degradation (0-th order). This criterion only uses tensile force components.

The critical bond stretch s_C for a microbrittle material is related to energy release rate G_{0C} and the discretization by means of the horizon.

The definition of the critical stretch is dependent on the model dimension and the peridynamic formulation. Since, currently, solely 3D-models are implemented in Peridigm only the respective equations are given here. For the equations of other model dimensions check the Peridigm Models and Verification Guide. Different equations exist for the state-based peridynamics and the special case of the bond-based peridynamics.

Beware, the following equations are based on the Griffith theory which assumes a pre-existing crack. If and how these relationships are valid for crack initiation must be investigated seperately.

Bond-based material models

The definition of the critical stretch from Table 2.3:

$$s_C = \sqrt{\frac{10 G_{0C}}{\pi c \delta^5}} \qquad = \sqrt{\frac{5 G_{0C}}{9 K \delta}} \tag{3.1}$$

With the bulk modulus K, the horizon δ and the micromodulus or bond constant c from Table 2.3.

State-based material models

The definition of the critical stretch from Table 2.3:

$$s_C = \sqrt{\frac{G_{0C}}{\left[3G + \left(\frac{3}{4}\right)^4 \left(K - \frac{5G}{3}\right)\right] \delta}} \tag{3.2}$$

3.4.2.2. Literature

- [7]

3.4.2.3. Model sketch

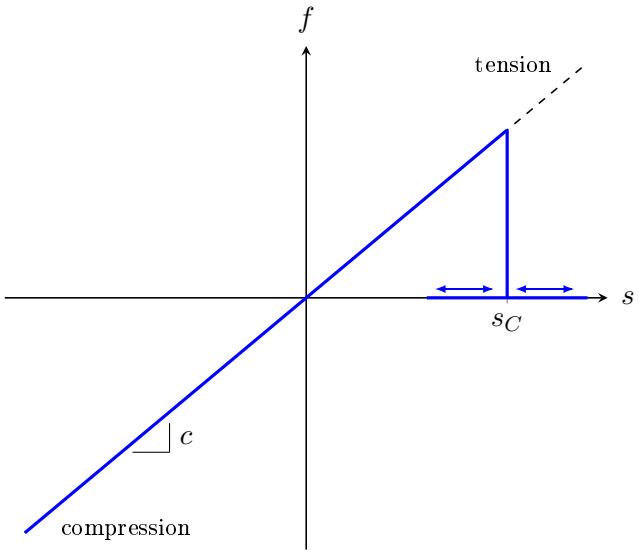


Figure 3.16.: 0-th order critical stretch model

3.4.2.4. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- /src/damage/Peridigm_CriticalStretchDamageModel.cpp
- /src/damage/Peridigm_CriticalStretchDamageModel.hpp

3.4.2.5. Input parameters

List

Name	Type	Required	Default	Description
Damage Model	string	✓	-	Damage model "Critical Stretch"
Critical Stretch[1]	double> 0.0	✓	-	Critical bond stretch
Thermal Expansion Coefficient[2,3]	double	-	-	

Remarks

1. The critical bond stretch is dependent of the horizon δ. Therefore, it is no pure material parameter.
2. If the thermal expansion coefficient parameter is omitted, deformations resulting from thermal expansion are not considered in the bond damage calculation.
3. To get consistent results, the value should correspond to the coefficient of thermal expansion defined in the respective material model used in the block definition. To avoid double definition, the use of a variable is proposed, see 2.2.4.3.
4. The criterion only applies for bonds in tension, as the relative extension of a bond is compared to the critical stretch value, see method `computeDamage()` in `Peridigm_CriticalStretchDamageModel.cpp`. In compression the relative extension of a bond is negative and can therefore never reach the positive critical stretch value.
5. The critical stretch model is faster compared to the interface aware model, but no check if the a point is completely debonded is included. See section 3.4.4 instead.

3.4.2.6. Exemplary input section

XML-format

without thermal expansion:

```
<ParameterList name="Damage Models">
  <ParameterList name="My Critical Stretch Damage Model">
    <Parameter name="Damage Model" type="string" value="Critical Stretch
    "/>
    <Parameter name="Critical Stretch" type="double" value="0.001"/>
  </ParameterList>
</ParameterList>
```

with thermal expansion (from `ThermalExpansionBondFailure.xml`):

```
<ParameterList name="Damage Models">
  <ParameterList name="My Critical Stretch Damage Model">
    <Parameter name="Damage Model" type="string" value="Critical Stretch
    "/>
    <Parameter name="Critical Stretch" type="double" value="0.05"/>
    <Parameter name="Thermal Expansion Coefficient" type="double" value
    ="10.0e-6"/>
  </ParameterList>
</ParameterList>
```

Free format

```
Damage Models
  My Damage Model
    Damage Model "Critical Stretch"
    Critical Stretch 0.001
```

YAML-format

-

3.4.2.7. List of examples

- From `examples/`:
  - `disk_impact/disc_impact.peridigm`
  - `fragmenting_cylinder/fragmenting_cylinder.peridigm`
- From `test/regression/`:
  - `Contact_Perforation/Contact_Perforation.xml`
  - `BondBreakingInitialVelocity/BondBreakingInitialVelocity.xml`
  - `ThermalExpansionBondFailure/ThermalExpansionBondFailure.xml`

3.4.3. Time Dependent Critical Stretch

3.4.3.1. Description

See section 3.4.2.1. The difference is, that a time-dependent function can be used to define the critical stretch instead of the absolute value from section 3.4.2.

3.4.3.2. Literature

See section 3.4.2.2.

3.4.3.3. Model sketch

See section 3.4.2.3.

3.4.3.4. Code

Release version

Available from the current master

Required compiler options

-

Routines

- from `src/damage/`:
  - `Peridigm_UserDefinedTimeDependentCriticalStretchDamageModel.cpp`
  - `Peridigm_UserDefinedTimeDependentCriticalStretchDamageModel.hpp`

3.4.3.5. Input parameters

List

Name	Type	Required	Default	Description
Damage Model	string	✓	-	Damage model "Time Dependent Critical Stretch"
Critical Stretch[1]	string	✓	-	A function defining a time-dependent value for the critical stretch
Thermal Expansion Coefficient[2,3]	double	-	-	

Remarks

1. The critical bond stretch is dependent of the horizon δ. Therefore, it is no pure material parameter.
2. If the thermal expansion coefficient parameter is omitted, deformations resulting from thermal expansion are not considered in the bond damage calculation.
3. To get consistent results, the value should correspond to the coefficient of thermal expansion defined in the respective material model used in the block definition. To avoid double definition, the use of a variable is proposed, see 2.2.4.3.
4. The criterion only applies for bonds in tension, as the relative extension of a bond is compared to the critical stretch value, see method `computeDamage()` in `Peridigm_UserDefinedTimeDependentCriticalStretchDamageModel.cpp`. In compression the relative extension of a bond is negative and can therefore never reach the positive critical stretch value.
5. The critical stretch model is faster compared to the interface aware model, but does not work for correspondence material models. See section 3.4.4 instead.

3.4.3.6. Exemplary input section

XML-format

from `BondBreakingInitialVelocity.xml`

```
<ParameterList name="Damage Models">
  <ParameterList name="My Critical Stretch Damage Model">
    <Parameter name="Damage Model" type="string" value="Time Dependent
   Critical Stretch"/>
    <Parameter name="Time Dependent Critical Stretch" type="string" value
   =" if(t &lt;= 7.0e-5)\{ value = 50.0; \} else\{ value = 0.015; \} "/>
```

```
  </ParameterList>
</ParameterList>
```

Free format

-

YAML-format

-

3.4.3.7. List of examples

- From `test/verification/`:
  - `BondBreakingInitialVelocity_TimeDependentCS/BondBreakingInitialVelocity.xml`

3.4.4. Interface Aware

3.4.4.1. Description

The interface aware damage model is a critical stretch model (cf. section 3.4.2). The difference is in the implemented algorithm.

> In this first pass, any "rank deficient" nodes are taken out of the linear system by pushing them into a boundary condition node set: RANK_DEFICIENT_NODES. All the bonds are broken for these nodes so it effectively removes them from the analysis. A new damage model is used to do the tasks above: Interface Aware. This model does not allow bonds to break between blocks if one of the blocks does not have a damage model. This damage model also does the checking of nodes if they meet the rank deficient criteria. A flag was also introduced that enables the user to turn off solver heuristics. For the examples tested with this commit, none of the solver heuristics were needed to get convergence. Examples have been tested that have a crack propagating through the entire domain in a single step and the solver manages to converge (if the Interface Aware damage model is used). To activate this flag, set "Disable Heuristics" to true in the solver parameters.

For the interface aware algorithm it is checked if each point has at least three bonds. If this is not the case, the point is completely disconnected. Therefore, this damage model is slower compared to the critical stretch model. However, for **correspondence material** this check is needed.

3.4.4.2. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- `/src/damage/Peridigm_InterfaceAwareDamageModel.cpp`
- `/src/damage/Peridigm_InterfaceAwareDamageModel.hpp`

3.4.4.3. Input parameters

List

Name	Type	Required	Default	Description
Damage Model	string	✓	-	Damage model "Interface Aware"
Critical Stretch[1]	double> 0.0	✓	-	Critical bond stretch
Coefficient[2,3]	double	-	-	

Remarks

1. The critical bond stretch is dependent of the horizon δ. Therefore, it is no pure material parameter.
2. If the thermal expansion coefficient parameter is omitted, deformations resulting from thermal expansion are not considered in the bond damage calculation.
3. To get consistent results, the value should correspond to the coefficient of thermal expansion defined in the respective material model used in the block definition. To avoid double definition, the use of a variable is proposed, see 2.2.4.3.
4. The criterion only applies for bonds in tension, as the relative extension of a bond is compared to the critical stretch value, see method `computeDamage()` in `Peridigm_CriticalStretchDamageModel.cpp`. In compression the relative extension of a bond is negative and can therefore never reach the positive critical stretch value.
5. The critical stretch model (3.4.2) is faster compared to the interface aware model. The interface aware model includes a check if a point is completely debonded.
6. Modeling damages with correspondence material a fine discretization is needed.

3.4.4.4. Exemplary input section

XML-format

-

Free format

```
Damage Models
  My Damage Model
    Damage Model "Interface Aware"
    Critical Stretch 0.001
```

YAML-format

-

3.4.4.5. List of examples

-

3.4.5. Critical Energy

3.4.5.1. Description

Multiple damage criteria based on the critical energy density of a bond are implemented. The model works for isotropic ordinary elastic material only.

3.4.5.2. Literature

[28]: Original energy release damage model proposed by Foster et al.
[29]: Extended energy release damage model proposed by Willberg et al., including the work of Foster et al.

3.4.5.3. Model sketch

-

3.4.5.4. Code

Release version

Not released yet. Under development.

Required compiler options

-

Routines

- Synchronization between cores:
  - `/src/damage/Peridigm.cpp`
  - `/src/damage/Peridigm.hpp`
  - `/src/damage/Peridigm_ModelEvaluator.cpp`
  - `/src/damage/Peridigm_ModelEvaluator.hpp`
  - `/src/damage/Peridigm_ElasticMaterial.cpp`
  - `/src/damage/Peridigm_Material.cpp`
  - `/src/damage/Peridigm_Material.hpp`
- Damage computation:
  - `/src/damage/Peridigm_EnergyReleaseDamageModel.cpp`
  - `/src/damage/Peridigm_EnergyReleaseDamageModel.hpp`

Details of implementation are given in the Peridigm_Development_Guide.

3.4.5.5. Input parameters

List

Mode-independent model based on [28]

Name	Type	Required	Default	Description
Damage Model	string	✓	-	Damage model "Critical Energy"
Critical Energy Tension[2,3]	double> 0.0	✓	-	Critical energy release rate G_{0C} [28]
Type[4]	string	✓	-	"Energy Criterion"

Mode-dependent model based on [29]

Name	Type	Required	Default	Description
Damage Model	string	✓	-	"Critical Energy"
Critical Energy Tension[2,3]	double> 0.0	✓	-	Critical energy release rate G_{IC} [28] for tension
Critical Energy Compression[2,3]	double> 0.0	-	-	Critical energy release rate G_{IC} [28] for compression
Critical Energy Shear[2,3]	double> 0.0	-	-	Critical energy release rate G_{IIC} [28]
Type[4]	string	✓	Energy Criterion	Interaction between Mode I and II: "Energy Criterion" | "Power Law" | "Separated"

Remarks

1. The critical energy model works only for "Elastic" material, see section 3.3.2.
2. All critical energies are independent of the horizon δ.
3. If the thermal expansion coefficient parameter is omitted, deformations resulting from thermal expansion are not considered in the bond damage calculation.
4. Interaction types:

   Energy Criterion: The criterion according [28] is applied, with the assumption that the stated value for "Critical Energy Tension" corresponds to the total critical energy release rate instead of only the mode I portion.

   Power Law: Mode I and II interaction by means of a quadratic power law:

   $$\left(\frac{G_I}{G_{IC}}\right)^2 + \left(\frac{G_{II}}{G_{IIC}}\right)^2 = 1$$

   where $G_{0C} = G_{IC} + G_{IIC}$

   Separated: No interaction between modes. Individual energies are checked separately.

   Only energies defined in the input deck are checked.

3.4.5.6. Exemplary input section

XML-format

Mode-independent model based on [28]

```
<ParameterList name="Damage Models">
  <ParameterList name="My Critical Energy Damage Model">
    <Parameter name="Damage Model" type="string" value="Critical Energy
    "/>
    <Parameter name="Critical  Energy Tension" type="double" value
    ="0.001"/>
    <Parameter name="Type" type="string" value="Energy Criterion"/>
  </ParameterList>
</ParameterList>
```

Mode-dependent model based on [29]

Type: quadratic power law

```
<ParameterList name="Damage Models">
  <ParameterList name="My Critical Energy Damage Model">
    <Parameter name="Damage Model" type="string" value="Critical Energy
   "/>
    <Parameter name="Critical  Energy Tension" type="double" value
   ="0.001"/>
    <Parameter name="Critical  Energy Compression" type="double" value
   ="0.001"/>
    <Parameter name="Critical  Energy Shear" type="double" value
   ="0.001"/>
    <Parameter name="Type" type="string" value="Power Law"/>
  </ParameterList>
</ParameterList>
```

Type: check all values separately

```
<ParameterList name="Damage Models">
  <ParameterList name="My Critical Energy Damage Model">
    <Parameter name="Damage Model" type="string" value="Critical Energy
   "/>
    <Parameter name="Critical  Energy Tension" type="double" value
   ="0.001"/>
    <Parameter name="Critical  Energy Compression" type="double" value
   ="0.001"/>
    <Parameter name="Critical  Energy Shear" type="double" value
   ="0.001"/>
    <Parameter name="Type" type="string" value="Separated"/>
  </ParameterList>
</ParameterList>
```

Free format

Mode-independent model based on [28]

```
Damage Models
  My Damage Model
    Damage Model "Critical Energy"
    Critical Energy Tension 0.001
    Type "Energy Criterion"
```

Mode-dependent model based on [29]

Type: quadratic power law

```
Damage Models
  My Damage Model
    Damage Model "Critical Energy"
    Critical Energy Tension 0.001
    Critical Energy Compression 0.001
    Critical Energy Shear 0.001
    Type "Power Law"
```

Type: check all values separately

```
Damage Models
  My Damage Model
    Damage Model "Critical Energy"
    Critical Energy Tension 0.001
    Critical Energy Compression 0.001
    Critical Energy Shear 0.001
    Type "Separated"
```

YAML-format

-

3.4.5.7. List of examples

Added to PeriDoX.

- From `Models/`:
  - `Models/DCB`

3.5. Properties

3.5.1. Blocks

3.5.1.1. Description

Peridynamic property/section definition.

3.5.1.2. Code

- /src/core/Peridigm_Block.cpp.cpp
- /src/core/Peridigm_Block.cpp.hpp
- /src/core/Peridigm_BlockBase.cpp
- /src/core/Peridigm_BlockBase.cpp

3.5.1.3. Input parameters

List

Name	Type	Required	Default	Description
Block Names	string	✓	-	Block names seperated by spaces
Material	string	✓	-	Name of block material
Damage Model	string	-	-	Name of block damage model
Horizon	double	✓[1]	-	Block horizon

Remarks

1. For numerical reasons it is a good idea to choose units such that the horizon is roughly of order one (from the Peridigm Mailing List, by D.J. Littlewood, 07.07.17)

3.5.1.4. Exemplary input section

XML-format

Definition of one block:

```
<ParameterList name="Blocks">
  <ParameterList name="My Group of Blocks">
    <Parameter name="Block Names" type="string" value="block_1"/>
    <Parameter name="Material" type="string" value="My Elastic Material
    "/>
```

```
    <Parameter name="Horizon" type="double" value="1.75"/>
  </ParameterList>
</ParameterList>
```

Definition of multiple blocks with the same parameters:

```
<ParameterList name="Blocks">
  <ParameterList name="My Group">
    <Parameter name="Block Names" type="string" value="block_1 block_2"/>
    <Parameter name="Material" type="string" value="My Elastic Material
    "/>
    <Parameter name="Horizon" type="double" value="0.5025"/>
  </ParameterList>
</ParameterList>
```

Block definition with function parser, dependent on spatial position:

```
<ParameterList name="Group of Blocks E">
  <Parameter name="Block Names" type="string" value="block_5"/>
  <Parameter name="Material" type="string" value="My Material"/>
  <Parameter name="Horizon" type="string" value="0.5000001 + 0.2500001*(x
    -1.625) + 0.2500001*(y-1.625) + 0.2500001*(z+0.375)"/>
</ParameterList>
```

Free format

```
Blocks
  My Group of Blocks
    Block Names "block_1"
    Material "My Elastic Material"
    Horizon 2.0
```

YAML format

```
Blocks:
  My Block:
    Block Names: "block_1 block_2 block_3"
    Material: "My Material"
    Horizon: 0.751
```

3.5.1.5. List of examples

Basically all models in `/test/` and `/examples/`.

3.6. Boundary Conditions

3.6.1. Preliminaries

3.6.1.1. Types

The following boundary condition types are defined in Peridigm:

- Prescribed_Displacement
- Prescribed_Fluid_Pressure_U
- Prescribed_Temperature
- Initial_Temperature
- Initial_Displacement
- Initial_Velocity
- Initial_Fluid_Pressure_U
- Body_Force

They are parsed in `~\src\core\Peridigm_BoundaryAndInitialConditionManager.cpp`. The names are defined in `~\src\core\Peridigm_Enums.cpp`. The input of keywords seems to be case-insensitive.

In general only "Prescribed Displacement" can be applied in Peridigm for the application of homogeneous or inhomogeneous kinematic boundary conditions. There is no discrete command for prescribed velocities or accelerations, despite "Initial Velocity". However, using the function parser it is still possible to apply these kinds of boundary conditions using the time variable t.

Generally it is possible to apply any kind of velocity v or acceleration a boundary conditions by considering the relationship to the displacement u:

$$a(t) = \dot{v}(t) = \ddot{u}(t) \tag{3.3}$$

3.6.1.2. Application region

Usually the boundary conditions have to be applied to particular material points for example at the free end of the body. Therefore, a nodeset has to be defined in order to list the material points to which the boundary condition has to be applied to.

If an Exodus finite element mesh is used as a discretisation the nodesets might already be defined in the mesh file. However, it is possible to define new nodesets in the Peridigm input deck as well as it is required for the discretisation via text file.

In order to define nodesets for the discretisation via a .txt file there are two options described in the next two segments

Definition in individual text file

Description

A .txt file has to be generated which contains the numbers of the desired boundary material points from the discretisation .txt file in a column.

Let the .txt files "nodeset_1.txt" and "nodeset_2.txt" be given as

```
1
2
3
4
```

and

```
9
10
11
12
```

and define the Parameter names “Node Set One” and “Node Set Two” as

```
<Parameter name="Node Set One" type="string" value="nodeset_1.txt"/>
<Parameter name="Node Set Two" type="string" value="nodeset_2.txt"/>
```

Thus, the boundary condition can be applied to these particular material points by entering the name of the nodeset into the input deck section "Boundary Conditions".

XML format

```
<ParameterList name="Boundary Conditions">
  <Parameter name="Node Set One" type="string" value="nodeset_1.txt"/>
  <Parameter name="Node Set Two" type="string" value="nodeset_2.txt"/>
  <ParameterList name="Prescribed Displacement Min X Face">
    <Parameter name="Type" type="string" value="Prescribed Displacement
   "/>
    <Parameter name="Node Set" type="string" value="Node Set One"/>
    <Parameter name="Coordinate" type="string" value="x"/>
    <Parameter name="Value" type="string" value="0.0"/>
  </ParameterList>
  <ParameterList name="Prescribed Displacement Max X Face">
    <Parameter name="Type" type="string" value="Prescribed Displacement
   "/>
    <Parameter name="Node Set" type="string" value="Node Set Two"/>
    <Parameter name="Coordinate" type="string" value="x"/>
    <Parameter name="Value" type="string" value="-0.1*t/0.00005"/>
  </ParameterList>
</ParameterList>
```

Free format

In the free format, it is important, that the node set name ends with the term `Node Set`:

```
Boundary Conditions
  nset_NSET-BC_BOT_SIDE Node Set "nset_NSET-BC_BOT_SIDE.txt"
  Displacement-1-D-x
    Type "Prescribed Displacement"
    Node Set "nset_NSET-BC_BOT_SIDE Node Set"
    Coordinate "x"
    Value "0.0"
```

Definition inside the model file

The other possibility to define the nodeset is to list the number of the desired nodes explicitly, i. e. as

```
<Parameter name="Node Set Three" type="string" value="2 4"/>
<Parameter name="Node Set Four" type="string" value="1 2"/>
```

The value of `Parameter name` can now be used as shown in the previous example.

Free format example:

```
Boundary Conditions
  Min X Node Set "1"
  Max X Node Set "2"
  Initial Velocity Min X Face
    Type "Initial Velocity"
    Node Set "Min X Node Set"
    Coordinate "x"
    Value "1.0"
  Initial Velocity Max X Face
    Type "Initial Velocity"
    Node Set "Max X Node Set"
    Coordinate "x"
    Value "-1.0"
```

Remarks

1. Dependent on the chosen discretisation type, a base FE mesh or the direct specification of the peridynamic collocation points in a textfile, the application region is either the finite element node id or the peridynamic collocation point id.

2. In case you use a FE mesh as discretisation type and try to define a nodeset in the model file, the base mesh node numbers do not necessarily represent the peridynamic representation collocation point numbers

3.6.2. Initial Displacement

3.6.2.1. Description

Application of a displacement at time $t = 0$ to the application region. For $t > 0$ this boundary condition is inactive.

3.6.2.2. Code

- `/src/core/Peridigm_BoundaryAndInitialConditionManager.cpp`
- `/src/core/Peridigm_BoundaryAndInitialConditionManager.hpp`
- `/src/core/Peridigm_BoundaryCondition.cpp`
- `/src/core/Peridigm_BoundaryCondition.hpp`
- `/src/core/Peridigm_Enums.cpp`
- `/src/core/Peridigm_Enums.hpp`

3.6.2.3. Input parameters

List

Name	Type	Required	Default	Description
Type	string	✓	-	"Initial Displacement"
Node Set	string	✓[2]	-	Application region name | "Full Domain" | "All Sets"
Coordinate	string	✓	-	"x" | "y" | "z"
Value	string	✓[3]	-	String with function for function parser

Remarks

1. It is not totally clear what happens when a prescribed displacement and an initial displacement/velocity are applied to the same application region. Who wins?. See `BoundaryCondition.cpp` lines 136-138.
2. Any string that is not "Full Domain" or "All Sets" will be considered a custom node set name in the model or mesh file, dependent of the discretization type.
3. The string in the variable *Value* should start with `value =` . If it does not, Peridigm will automatically add it for the function parser to work.

3.6.2.4. Exemplary input section

XML format

```
<ParameterList name="Initial Displacement Min X Face">
  <Parameter name="Type" type="string" value="Initial Displacement"/>
  <Parameter name="Node Set" type="string" value="Min X Node Set"/>
  <Parameter name="Coordinate" type="string" value="x"/>
  <Parameter name="Value" type="string" value="0.1"/>
</ParameterList>
```

Free format

-

YAML format

-

3.6.2.5. List of examples

- From `test/verification/`:
  - `CompressionInitDisp_2x1x1/CompressionInitDisp_2x1x1.xml`
  - `CompressionImplicitEssentialBC_2x1x1/CompressionImplicitEssentialBC_-2x1x1.xml`

3.6.3. Initial Velocity

3.6.3.1. Description

Application of a velocity at time $t = 0$ to the application region. For $t > 0$ this boundary condition is inactive.

3.6.3.2. Code

- `/src/core/Peridigm_BoundaryAndInitialConditionManager.cpp`
- `/src/core/Peridigm_BoundaryAndInitialConditionManager.hpp`
- `/src/core/Peridigm_BoundaryCondition.cpp`
- `/src/core/Peridigm_BoundaryCondition.hpp`
- `/src/core/Peridigm_Enums.cpp`
- `/src/core/Peridigm_Enums.hpp`

3.6.3.3. Input parameters

List

Name	Type	Required	Default	Description
Type	string	✓	-	"Initial Velocity"
Node Set	string	✓[2]	-	Application region name | "Full Domain" | "All Sets"
Coordinate	string	✓	-	"x" | "y" | "z"
Value	string	✓[3]	-	String with function for function parser

Remarks

1. It is not totally clear what happens when a prescribed displacement and an initial displacement/velocity are applied to the same application region. Who wins?. See `BoundaryCondition.cpp` lines 136-138.
2. Any string that is not "Full Domain" or "All Sets" will be considered a custom node set name in the model or mesh file, dependent of the discretization type.
3. The string in the variable *Value* should start with `value =` . If it does not, Peridigm will automatically add it for the function parser to work.

3.6.3.4. Exemplary input section

XML format

```
<ParameterList name="Boundary Conditions">
  <ParameterList name="Left Side Initial Velocity">
    <Parameter name="Type" type="string" value="Initial Velocity"/>
    <Parameter name="Node Set" type="string" value="nodelist_1"/>
    <Parameter name="Coordinate" type="string" value="x"/>
    <Parameter name="Value" type="string" value="-100.0"/>
  </ParameterList>
</ParameterList>
```

Free format

Using "Full domain" for the node set definition instead of a pre-defined node set name, from `examples/fragmenting_cylinder/fragmenting_cylinder.peridigm`:

```
Boundary Conditions
  Initial Velocity X
    Type "Initial Velocity"
    Node Set "Full Domain"
    Coordinate "x"
    Value "(200 - 50*((z/0.05)-1)^2)*cos(atan2(y,x))"
  Initial Velocity Y
    Type "Initial Velocity"
    Node Set "Full Domain"
    Coordinate "y"
    Value "(200 - 50*((z/0.05)-1)^2)*sin(atan2(y,x))"
  Initial Velocity Z
    Type "Initial Velocity"
    Node Set "Full Domain"
    Coordinate "z"
    Value "(100*((z/0.05)-1))"
```

YAML format

-

3.6.3.5. List of examples

- From `examples/`:
  - `fragmenting_cylinder/fragmenting_cylinder.peridigm`
- From `test/regression/`:
  - `WaveInBar/WaveInBar.xml`

3.6.4. Initial Temperature

3.6.4.1. Description

Application of a temperature at time $t = 0$ to the application region. For $t > 0$ this boundary condition is inactive.

3.6.4.2. Code

- from `src/core/`:
  - `Peridigm_BoundaryAndInitialConditionManager.cpp`
  - `Peridigm_BoundaryAndInitialConditionManager.hpp`
  - `Peridigm_BoundaryCondition.cpp`
  - `Peridigm_BoundaryCondition.hpp`
  - `Peridigm_Enums.cpp`
  - `Peridigm_Enums.hpp`

3.6.4.3. Input parameters

List

Name	Type	Required	Default	Description
Type	string	✓	-	"Initial Temperature"
Node Set	string	✓[1]	-	Application region name | "Full Domain" | "All Sets"
Value	string	✓[2]	-	String with function for function parser

Remarks

1. Any string that is not "Full Domain" or "All Sets" will be considered a custom node set name in the model or mesh file, dependent of the discretization type.
2. The string in the variable *Value* should start with `value =` . If it does not, Peridigm will automatically add it for the function parser to work.
3. This boundary conditions might not work with some of the material models in the current implementation.

3.6.4.4. Exemplary input section

XML format

-

Free format

-

YAML format

-

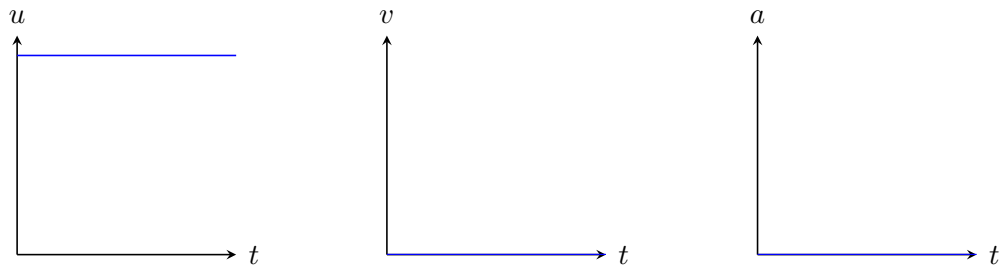
3.6.4.5. List of examples

3.6.5. Prescribed Displacement

3.6.5.1. Description

Definition of a prescribed displacement on a nodeset. The prescribed displacement might be dependent of the scalar coordinates of the nodeset members and/or the time. Thus, displacement and acceleration boundary conditions can be modeled with this card as well.

3.6.5.2. Sketch



3.6.5.3. Code

- from `src/core`:
  - `Peridigm_BoundaryAndInitialConditionManager.cpp`
  - `Peridigm_BoundaryAndInitialConditionManager.hpp`
  - `Peridigm_BoundaryCondition.cpp`
  - `Peridigm_BoundaryCondition.hpp`
  - `Peridigm_Enums.cpp`
  - `Peridigm_Enums.hpp`

3.6.5.4. Input parameters

List

Name	Type	Required	Default	Description
Type	string	✓	-	"Prescribed Displacement"
Node Set	string	✓[1]	-	Application region name | "Full Domain" | "All Sets"
Coordinate	string	✓	-	"x" | "y" | "z"
Value	string	✓[2]	-	String with function for function parser

Remarks

1. Any string that is not "Full Domain" or "All Sets" will be considered a custom node set name in the model or mesh file, dependent of the discretization type.
2. The string in the variable *Value* should start with `value =` . If it does not, Peridigm will automatically add it for the function parser to work.

3.6.5.5. Exemplary input section

XML format

```
<ParameterList name="Boundary Conditions">
  <ParameterList name="Prescr. Displacement Suppress Rigid Body Modes 1">
    <Parameter name="Type" type="string" value="Prescribed Displacement"/>
    <Parameter name="Node Set" type="string" value="nodelist_3"/>
    <Parameter name="Coordinate" type="string" value="y"/>
    <Parameter name="Value" type="string" value="0.0"/>
  </ParameterList>
  <ParameterList name="Prescr. Displacement Suppress Rigid Body Modes 2">
    <Parameter name="Type" type="string" value="Prescribed Displacement"/>
    <Parameter name="Node Set" type="string" value="nodelist_5"/>
    <Parameter name="Coordinate" type="string" value="y"/>
    <Parameter name="Value" type="string" value="0.0"/>
  </ParameterList>
  <ParameterList name="Prescr. Displacement Suppress Rigid Body Modes 3">
    <Parameter name="Type" type="string" value="Prescribed Displacement"/>
    <Parameter name="Node Set" type="string" value="nodelist_4"/>
    <Parameter name="Coordinate" type="string" value="z"/>
    <Parameter name="Value" type="string" value="0.0"/>
  </ParameterList>
  <ParameterList name="Prescr. Displacement Suppress Rigid Body Modes 4">
    <Parameter name="Type" type="string" value="Prescribed Displacement"/>
    <Parameter name="Node Set" type="string" value="nodelist_6"/>
    <Parameter name="Coordinate" type="string" value="z"/>
    <Parameter name="Value" type="string" value="0.0"/>
  </ParameterList>
</ParameterList>
```

Free format

```
Boundary Conditions
  Prescribed Displacement Fix Bottom Rigid Body Motion In X
```

```
    Type "Prescribed Displacement"
    Node Set "nodelist_3"
    Coordinate "x"
    Value "0.0"
  Prescribed Displacement Fix Bottom Rigid Body Motion In Z
    Type "Prescribed Displacement"
    Node Set "nodelist_4"
    Coordinate "z"
    Value "0.0"
  Prescribed Displacement Fix Top Rigid Body Motion In X
    Type "Prescribed Displacement"
    Node Set "nodelist_5"
    Coordinate "x"
    Value "0.0"
  Prescribed Displacement Fix Top Rigid Body Motion In Z
    Type "Prescribed Displacement"
    Node Set "nodelist_6"
    Coordinate "z"
    Value "0.0"
```

YAML format

```
Boundary Conditions:
  My Prescribed Displacement:
    Type: "Prescribed Displacement"
    Node Set: "My Node Set"
    Coordinate: "x"
    Value: "value = 5.0"
```

3.6.5.6. List of examples

- From `examples/`:
  - `tensile_test/tensile_test.peridigm`
- From `test/regression/`:
  - `Bar_OneBlock_OneMaterial_QS/Bar.xml`

3.6.6. Prescribed Velocity

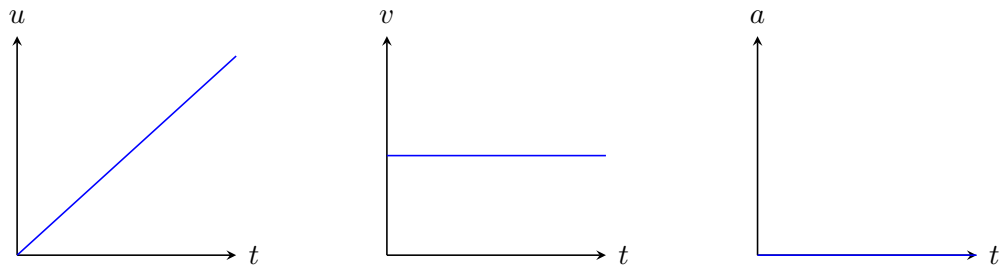
3.6.6.1. Description

Prescribed velocity is not available as an individual keyword. The behavior can be reproduced using the keyword `Prescribed Displacement` from subsection 3.6.5 by application of Equation 3.3 in combination with the function parser and the time variable t.

3.6.6.2. Types & Sketches

Constant velocity over whole time

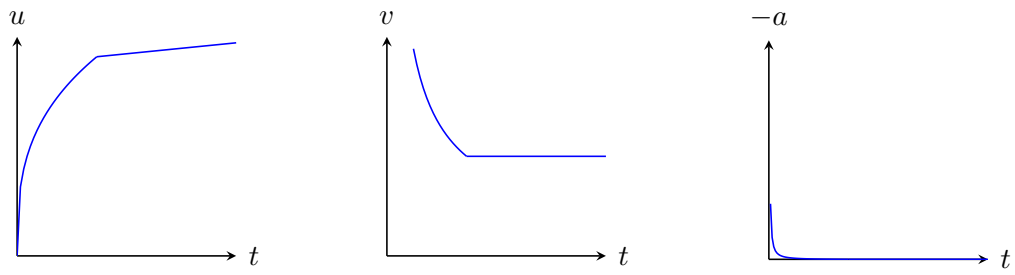
```
Boundary Conditions:
  My Prescribed Velocity:
    Type: "Prescribed Displacement"
    Node Set: "My Node Set"
    Coordinate: "x"
    Value: "value = 5.0*t"
```



Constant velocity after initial ramp

```
#{UI=1}           # Displacement right before initial damage
#{TI=1}           # Time right before initial damage
#{VINFTY=1}       # Constant velocity during damage initiation

Boundary Conditions
  My Prescribed Velocity
    Type: "Prescribed Displacement"
    Node Set: "My Node Set"
    Coordinate: "x"
    Value: " if(t <= {TI})\{ value = {UI}*{TI}^{-1/3}*t^(1/3); \} else\{
   value = {VINFTY}*(t-{TI})+{UI}; \} "
```



3.6.6.3. Code

See subsection 3.6.5

3.6.6.4. Input parameters

See subsection 3.6.5

3.6.6.5. Exemplary input section

XML format

```
<ParameterList name="Boundary Conditions">
  <ParameterList name="Prescribed Displacement Left Side">
    <Parameter name="Type" type="string" value="Prescribed Displacement"/>
    <Parameter name="Node Set" type="string" value="nodelist_1"/>
    <Parameter name="Coordinate" type="string" value="x"/>
    <Parameter name="Value" type="string" value="-0.01*t"/>
   </ParameterList>
  <ParameterList name="Prescribed Displacement Right Side">
    <Parameter name="Type" type="string" value="Prescribed Displacement"/>
    <Parameter name="Node Set" type="string" value="nodelist_2"/>
    <Parameter name="Coordinate" type="string" value="x"/>
    <Parameter name="Value" type="string" value="0.01*t"/>
   </ParameterList>
</ParameterList>
```

Free format

```
Boundary Conditions
  Prescribed Displacement Bottom
    Type "Prescribed Displacement"
    Node Set "nodelist_1"
    Coordinate "y"
    Value "y*0.005*t"
  Prescribed Displacement Top
    Type "Prescribed Displacement"
    Node Set "nodelist_2"
    Coordinate "y"
    Value "y*0.005*t"
```

YAML format

-

3.6.6.6. List of examples

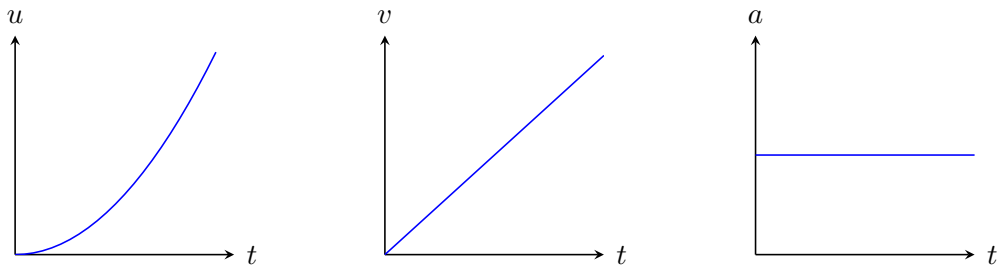
- From `examples/`:
  - `examples/tensile_test/tensile_test.peridigm`
- From `test/regression/`:
  - `Bar_OneBlock_OneMaterial_QS/Bar.xml`

3.6.7. Prescribed Acceleration

3.6.7.1. Description

Prescribed acceleration is not available as an individual keyword. The behavior can be reproduced using the keyword `Prescribed Displacement` from subsection 3.6.5 by application of Equation 3.3 in combination with the function parser and the time variable t.

3.6.7.2. Sketch



3.6.7.3. Code

See subsection 3.6.5

3.6.7.4. Input parameters

See subsection 3.6.5

3.6.7.5. Exemplary input section

XML format

-

Free format

-

YAML format

```
Boundary Conditions:
  My Prescribed Acceleration:
    Type: "Prescribed Displacement"
    Node Set: "My Node Set"
    Coordinate: "x"
    Value: "value = 5.0*t^2"
```

3.6.7.6. List of examples

3.6.8. Prescribed Temperature

3.6.8.1. Description

Definition of a prescribed temperature on a nodeset. The prescribed temperature might be dependent of the scalar coordinates of the nodeset members and/or the time.

3.6.8.2. Code

- from `src/core/`:
  - `Peridigm_BoundaryAndInitialConditionManager.cpp`
  - `Peridigm_BoundaryAndInitialConditionManager.hpp`
  - `Peridigm_BoundaryCondition.cpp`
  - `Peridigm_BoundaryCondition.hpp`
  - `Peridigm_Enums.cpp`
  - `Peridigm_Enums.hpp`

3.6.8.3. Input parameters

List

Name	Type	Required	Default	Description
Type	string	✓	-	"Prescribed Temperature"
Node Set	string	✓[1]	-	Application region name | "Full Domain" | "All Sets"
Value	string	✓	-	String with function for function parser

Remarks

1. Any string that is not "Full Domain" or "All Sets" will be considered a custom node set name in the model or mesh file, dependent of the discretization type.
2. The string in the variable *Value* should start with `value =` . If it does not, Peridigm will automatically add it for the function parser to work.
3. This boundary conditions might not work with some of the material models in the current implementation.

3.6.8.4. Exemplary input section

XML format

```
<ParameterList name="Boundary Conditions">
  <ParameterList name="Prescribed Thermal Loading">
    <Parameter name="Type" type="string" value="Prescribed Temperature"/>
    <Parameter name="Node Set" type="string" value="FULL_domain"/>
    <Parameter name="Value" type="string" value="10000.0 - 10000.0*0.5*(
   cos(3.14159265359 + t*3.14159265359/0.1) + 1.0)"/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

3.6.8.5. List of examples

- From `test/verification/`:
  - `ThermalExpansionBondFailure/ThermalExpansionBondFailure.xml`
  - `ThermalExpansionBondFailure/ThermalExpansionCube.xml`

3.6.9. Body Force

3.6.9.1. Description

Application of a volumetric load to an application region.

In contrast to continuum mechanical models the peridynamic model works with the so called *force density*. The force density F_V is defined as force F per volume V by $F_V = F/V$. The dual or pairwise force density function as an internal variable is then defined as force per volume squared (see page 14 in [30]). External forces on the peridynamic collocation points are expressed in the bond force density. The bond force density has the unit force per unit squared as well. Thus, the nodal, surface and volume loads from continuum mechanics can easily be transformed using the application region volume.

However, one has to be consider that the force density is applied for the whole volume. The resulting displacements are given at the points. Therefore, there are small discrepancies in the position of the applied load in peridynamics compared to the discretized continuum mechanics model.

3.6.9.2. Literature

- [30]

3.6.9.3. Code

- from `src/core/`:
  - `Peridigm_BoundaryAndInitialConditionManager.cpp`
  - `Peridigm_BoundaryAndInitialConditionManager.hpp`
  - `Peridigm_BoundaryCondition.cpp`
  - `Peridigm_BoundaryCondition.hpp`
  - `Peridigm_Enums.cpp`
  - `Peridigm_Enums.hpp`

3.6.9.4. Input parameters

List

Name	Type	Required	Default	Description
Type	string	✓	-	"Body Force"
Node Set	string	✓[1]	-	Application region name | "Full Domain" | "All Sets"
Coordinate	string	✓	-	"x" | "y" | "z"
Value[2]	string	✓	-	String with function for function parser

Remarks

1. Any string that is not "Full Domain" or "All Sets" will be considered a custom node set name in the model or mesh file, dependent of the discretization type.
2. The string in the variable *Value* should start with `value =` . If it does not, Peridigm will automatically add it for the function parser to work.

3.6.9.5. Exemplary input section

XML format

```
<ParameterList name="Applied Loading X">
  <Parameter name="Type" value="Body Force" type="string"/>
  <Parameter name="Node Set" value="nodelist_1" type="string"/>
  <Parameter name="Coordinate" value="x" type="string"/>
  <Parameter name="Value" value="1E-5 * 94.25^2 * x" type="string"/>
</ParameterList>

<ParameterList name="Applied Loading Z">
  <Parameter name="Type" value="Body Force" type="string"/>
  <Parameter name="Node Set" value="nodelist_1" type="string"/>
  <Parameter name="Coordinate" value="y" type="string"/>
  <Parameter name="Value" value="1E-5 * 94.25^2 * y" type="string"/>
</ParameterList>
```

Free format

-

YAML format

-

3.6.9.6. List of examples

- From `test/regression/`:
  - `Body_Force/Body_Force_*.xml`
  - `CentrifugalLoad/CentrifugalLoad.xml`
  - `Multiphysics_QS_3x2x2/Multiphysics_QS_3x2x2.xml`

3.7. Contact

3.7.1. General options

3.7.1.1. Description

Define contact for the model. Contact forces in Peridigm are not applied for nodes that are bonded. This avoids the situation where two nodes interact via both contact and the constitutive model.

- From David Littlewood:
  - Be aware also that the input deck syntax for contact is not representative of what's actually in the code. [...] As far as I remember, what's actually implemented in Peridigm is a all-to-all contact model that is active for any pair of nodes that are not bonded to each other. So, there is no consideration of blocks, node sets, etc., and basically both self contact and general contact are always on.
  - The portion of the input deck in which contact interactions are specified is not currently functional. Contact only operates in two modes: either completely off, or on for everything. By "on for everything" I mean that every node interacts with every other node, regardless of material block or what is specified in the input deck. This is obviously not ideal and is confusing to users. The contact portion of the code underwent a significant refactor a few years ago. A good deal of progress was made with respect to performance and I/O, but unfortunately the code ended up in a partially refactored state. One part that was not completed is user-defined per-block contact interactions. The parsing was set up but the internal logic was not implemented. The contact code in general needs significant work IMHO.
- From James O'Grady:
  Nodes that are initially bonded will never interact via the contact model, even after the bond between them is broken. I'm not sure whether that includes bonds broken by a pre-crack.

3.7.1.2. Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- /src/core/Peridigm_ContactManager.cpp
- /src/core/Peridigm_ContactManager.hpp

3.7.1.3. Input parameters

List

Name	Type	Required	Default	Description
Search Radius	double	✓	0.0	Radius for contact search
Search Frequency	int	✓	0	Contact rebalancing frequency in steps
Models	list	✓		Contact models, see section 3.7.2
Interactions	list	✓		List defining the interaction options, see section 3.7.3

Remarks

1. Contact is enabled only for the explicit solver (Verlet). In case any other solver is used, the contact definition has no effect.
2.

3.7.1.4. Exemplary input section

XML-format

All blocks contact:

```
<ParameterList name="Contact">
  <Parameter name="Verbose" type="bool" value="true"/>
  <Parameter name="Search Radius" type="double" value="0.08"/>
  <Parameter name="Search Frequency" type="int" value="1"/>
  <ParameterList name="Models">
    <ParameterList name="My Contact Model">
      <Parameter name="Contact Model" type="string" value="Short Range
   Force"/>
      <Parameter name="Contact Radius" type="double" value="0.08"/>
      <Parameter name="Spring Constant" type="double" value="2000.0e3"/>
    </ParameterList>
  </ParameterList>
  <ParameterList name="Interactions">
    <ParameterList name="General Contact">
```

```
      <Parameter name="Contact Model" type="string" value="My Contact
    Model"/>
     </ParameterList>
  </ParameterList>
</ParameterList>
```

Specific block contact:

```
<ParameterList name="Contact">
  <Parameter name="Search Radius" type="double" value="0.6"/>
  <Parameter name="Search Frequency" type="int" value="50"/>
  <ParameterList name="Models">
    <ParameterList name="My Contact Model">
      <Parameter name="Contact Model" type="string" value="Short Range
    Force"/>
      <Parameter name="Contact Radius" type="double" value="0.2"/>
      <Parameter name="Spring Constant" type="double" value="1950.0e3"/>
    </ParameterList>
  </ParameterList>
  <ParameterList name="Interactions">
    <ParameterList name="Interaction Projectile with Target">
      <Parameter name="First Block" type="string" value="block_1"/>
      <Parameter name="Second Block" type="string" value="block_2"/>
        <Parameter name="Contact Model" type="string" value="My Contact
    Model"/>
    </ParameterList>
  </ParameterList>
</ParameterList>
```

Free format

```
Contact
  Verbose "true"
  Search Radius {0.8*MODEL_ESIZE_FIBRE}
  Search Frequency 1
  Models
    My Contact Model
      Contact Model "Short Range Force"
      Contact Radius {0.8*MODEL_ESIZE_FIBRE}
      Spring Constant 2000.001e3
  Interactions
    General Contact
      Contact Model "My Contact Model"
```

YAML format

-

3.7.1.5. List of examples

- From `test/regression/`:
  - `Contact_Cubes/Contact_Cubes.xml`
  - `Contact_Cubes_Interaction_Blocks/Contact_Cubes_Interaction_Blocks.xml`
  - `Contact_Perforation/Contact_Perforation.xml`
  - `Contact_Perforation_With_Restart/Contact_Perforation_With_Restart.xml`
  - `Contact_Ring/Contact_Ring.xml`
- From `test/verification/`:
  - `Contact_2x1x1/Contact_2x1x1.xml`
  - `Contact_Friction/Contact_Friction.xml`
  - `Contact_Friction_Time_Dependent_Coefficient/Contact_Friction_Time_-Dependent_Coefficient.xml`

3.7.2. Models

3.7.2.1. Short Range Force

Description

A simple contact model for the interaction of non-bonded elements.

- From David Littlewood:
  To the best of my knowledge, the short-range contact model in Peridigm is completely ad hoc. It isn't tied to anything physical as far as I know, and I'm not aware of any general usage rules. It's just a repulsive force that was implemented to give somewhat reasonable results for impact problems.

Literature

- [7]
- [2]
- [31] & [16]

Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

- `/src/contact/Peridigm_ShortRangeForceContactModel.cpp`
- `/src/contact/Peridigm_ShortRangeForceContactModel.hpp`

Input parameters

List

Name	Type	Required	Default	Description
Contact Model	string	✓	-	Contact model type "Short Range Force"
Contact Radius[2]	double	✓	-	
Spring Constant[3]	double	✓	-	
Friction Coefficient[4]	double	-	0.0	
Horizon[5]	double	-	0.0	

Remarks

1. Contact algorithm operates on planar facets, these are created from peridynamic collocation points, see [31] & [16]
2. Initial guess for contact radius: somewhat smaller than the initial node spacing, so that it doesn't add forces to material in the undeformed or slightly deformed configuration. I would also expect it to be large enough to prevent material interpenetration.
3. From David Littlewood: The spring force parameter is determined by trial and error, although a good initial guess might be 10 times the force of the constitutive model.
4. In case `Friction Coefficient` is not defined, no friction is applied by using the default value
5.

Exemplary input section

XML-format

without friction:

```
<ParameterList name="Models">
  <ParameterList name="My Contact Model">
    <Parameter name="Contact Model" type="string" value="Short Range
   Force"/>
    <Parameter name="Contact Radius" type="double" value="0.2"/>
    <Parameter name="Spring Constant" type="double" value="1950.0e3"/>
  </ParameterList>
</ParameterList>
```

with friction:

```
<ParameterList name="Models">
  <ParameterList name="My Contact Model">
    <Parameter name="Contact Model" type="string" value="Short Range
   Force"/>
    <Parameter name="Contact Radius" type="double" value="0.2"/>
    <Parameter name="Spring Constant" type="double" value="1.0e12"/>
    <Parameter name="Friction Coefficient" type="double" value="0.3"/>
  </ParameterList>
</ParameterList>
```

Free format

Without friction:

```
Models
  My Contact Model
    Contact Model "Short Range Force"
    Contact Radius {0.8*MODEL_ESIZE_FIBRE}
    Spring Constant {10.0001*MAT_FIBRE_MODULUS_YOUNG}
```

YAML format

-

List of examples

- From `test/verification/`:
  - `Contact_Friction/Contact_Friction.xml`

3.7.2.2. Time Dependent Short Range Force

Description

A simple contact model for the interaction of non-bonded elements.

Literature

- [7]
- [2]

Code

Release version

Available from the version after 1.4.1.

Required compiler options

-

Routines

- from `src/contact/`:
  - `Peridigm_UserDefinedTimeDependentShortRangeForceContactModel.cpp`
  - `Peridigm_UserDefinedTimeDependentShortRangeForceContactModel.hpp`

Input parameters

List

Name	Type	Required	Default	Description
Contact Model	string	✓	-	Contact model type "Time Dependent Short Range Force"
Contact Radius	double	✓	-	
Spring Constant	double	✓	-	
Friction Coefficient[1]	double	-	0.0	
Horizon	double	-	0.0	

Remarks

1. In case `Friction Coefficient` is not defined, no friction is applied by using the default value

Exemplary input section

XML-format

with friction:

```
<ParameterList name="Models">
  <ParameterList name="My Contact Model">
    <Parameter name="Contact Model" type="string" value="Time Dependent
   Short Range Force"/>
    <Parameter name="Contact Radius" type="double" value="0.2"/>
    <Parameter name="Spring Constant" type="double" value="1.0e12"/>
    <Parameter name="Friction Coefficient" type="string" value=" if(t &lt
   ;= 4.00e-5)\{ value = 1.95; \} else\{ value = 0.3; \} "/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

List of examples

- From `test/verification/`:
  - `Contact_Friction_Time_Dependent_Coefficient/Contact_Friction_Time_-Dependent_Coefficient.xml`

3.7.3. Interactions

3.7.3.1. General Contact

Description

Everything is in contact with everything. This type of interaction is activated by setting the parameter list name to General Contact.

Literature

-

Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

-

Input parameters

List

Name	Type	Required	Default	Description
Contact Model	string	✓	-	Contact model type from section 3.7.2

Remarks

-

Exemplary input section

XML-format

```
<ParameterList name="Interactions">
  <ParameterList name="General Contact">
    <Parameter name="Contact Model" type="string" value="My Contact Model
    "/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

List of examples

- From `test/regression/`:
  - `Contact_Cubes/Contact_Cubes.xml`
  - `Contact_Ring/Contact_Ring.xml`
- From `test/verification/`:
  - `Contact_Friction/Contact_Friction.xml`

3.7.3.2. Self Contact

Description

Define self contact. This type of interaction is activated by setting the parameter list name to Self Contact.

Literature

-

Code

Release version

Available from version 1.2.

Required compiler options

-

Routines

-

Input parameters

List

Name	Type	Required	Default	Description
Contact Model	string	✓	-	Contact model type from section 3.7.2

Remarks

-

Exemplary input section

XML-format

-

Free format

-

YAML format

-

List of examples

-

3.7.3.3. User defined block interaction

Description

Define interactions between individual blocks. This type of interaction is activated by setting the parameter list name to any different name than General Contact and Self Contact.

Literature

-

Code

Release version

Available from version 1.4.

Required compiler options

-

Routines

-

Input parameters

List

Name	Type	Required	Default	Description
Contact Model	string	✓	-	Contact model type from section 3.7.2
First Block	string	✓	-	Block name/id of first interaction partner
Second Block	string	✓	-	Block name/id of second interaction partner

Remarks

-

Exemplary input section

XML-format

```
<ParameterList name="Interactions">
  <ParameterList name="My Contact Interaction">
    <Parameter name="Contact Model" type="string" value="My Contact Model
    "/>
    <Parameter name="First Block" type="string" value="block_2"/>
    <Parameter name="Second Block" type="string" value="block_3"/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

List of examples

- From `test/regression/`:
  - `Contact_Cubes_Interaction_Blocks/Contact_Cubes_Interaction_Blocks.xml`
  - `Contact_Perforation/Contact_Perforation.xml`
- From `test/verification/`:
  - `Contact_2x1x1/Contact_2x1x1.xml`
  - `Contact_Friction/Contact_Friction.xml`
  - `Contact_Friction_Time_Dependent_Coefficient/Contact_Friction_Time_-Dependent_Coefficient.xml`

3.8. Compute Class Parameters - User defined calculation data

3.8.1. Compute Class Parameters for node sets

3.8.1.1. Description

Define user defined calculation data for nodesets. These quantities can later be output, see section 3.11.3.

3.8.1.2. Code

- `/src/compute/Peridigm_Compute_Node_Set_Data.cpp`
- `/src/compute/Peridigm_Compute_Node_Set_Data.cpp`

3.8.1.3. Input parameters

List

Name	Type	Required	Default	Description
Compute Class	string	✓		"Node_Set_Data"
Calculation Type	string	✓	-	"Minimum" | "Maximum" | "Sum"
Node Set	string	✓	-	Node set name
Variable	string	✓[1,3]	-	Output variable
Output Label	string	✓[2]	-	Output label for output section

Remarks

1. The output value identifier are defined in `src/io/mesh_output/Field.h`. Possible values are defined in subsubsection 3.11.1.1.
2. For the output to have any effect, the output label must be defined in the "Output" section of the input deck.
3. The item requested as `Variable` must be defined and requested as a global `Output Variable` in the `Output` block, e.g.: In case you request a `Compute Class Parameter` *Top Reaction Force* with *Variable "Force"*:

```
Compute Class Parameters
  Top Reaction Force
    ...
    Variable "Force"
    Output Label "Top_Reaction_Force"
```

Than `Force "true"` must be defined in the `Output` section:

```
Output
  Output File Type "ExodusII"
  ..
  Output Variables
    Block_Id "false"
    ...
    Force "true"
    Top_Reaction_Force "true"
```

Without `Force "true"` an error is thrown as the group variables cannot be extracted from the global solution.

4. There seems to be a bug in the text file reader in case nodesets are defined by additional text files for the text file discretization from 2.3.2. There are two possible workarounds:
   a) Use the python script *text_to_genesis.py* to convert your text file into a genesis file. This script is in the scripts directory of the Peridigm repo, you probably will have to edit the *PATH* at the top of the python file to point to your Trilinos lib directory so it can load *exodus.py*.
   b) If possible, create an individual block for the domain of the nodeset and use the compute class parameter option for blocks from 3.8.2

3.8.1.4. Exemplary input section

XML-format

from `test/regression/Contact_Ring/Contact_Ring.xml`:

```
<ParameterList name="Compute Class Parameters">
  <ParameterList name="Node Set 10 Stored Elastic Energy">
    <Parameter name="Compute Class" type="string" value="Node_Set_Data"/>
    <Parameter name="Calculation Type" type="string" value="Sum"/>
    <Parameter name="Node Set" type="string" value="nodelist_10"/>
    <Parameter name="Variable" type="string" value="Stored_Elastic_Energy
   "/>
    <Parameter name="Output Label" type="string" value="
   NS_10_Stored_Elastic_Energy"/>
  </ParameterList>
  <ParameterList name="Node Set 11 Stored Elastic Energy">
    <Parameter name="Compute Class" type="string" value="Node_Set_Data"/>
    <Parameter name="Calculation Type" type="string" value="Sum"/>
    <Parameter name="Node Set" type="string" value="nodelist_11"/>
```

```
    <Parameter name="Variable" type="string" value="Stored_Elastic_Energy
    "/>
    <Parameter name="Output Label" type="string" value="
    NS_11_Stored_Elastic_Energy"/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

3.8.1.5. List of examples

- From `test/regression/`:
  - `Contact_Ring/Contact_Ring.xml`
  - `WaveInBar_MultiBlock/WaveInBar_MultiBlock.xml`

3.8.2. Compute Class Parameters for blocks

3.8.2.1. Description

User defined output for blocks using `Compute Class Parameters`. These quantities can later be output, see section 3.11.4.

3.8.2.2. Code

- `/src/compute/Peridigm_Compute_Block_Data.cpp`
- `/src/compute/Peridigm_Compute_Block_Data.hpp`

3.8.2.3. Input parameters

List

Name	Type	Required	Default	Description
Compute Class	string	✓		"Block_Data"
Calculation Type	string	✓	-	"Minimum" | "Maximum" | "Sum"
Block	string	✓	-	Block name
Variable	string	✓[1,3]	-	Output variable
Output Label	string	✓[2]	-	Output label for output section

Remarks

1. The output value identifier are defined in `src/io/mesh_output/Field.h`. Possible values are defined in subsubsection 3.11.1.1.
2. For the output to have any effect, the output label must be defined in the "Output" section of the input deck. This is necessary since the `Compute Class Parameters` are extracted from the global solution/output data.
3. The item requested as `Variable` must be defined and requested as a global `Output Variable` in the `Output` block, e.g.: In case you request a `Compute Class Parameter` *Top Reaction Force* with *Variable "Force"*:

```
Compute Class Parameters
  Top Reaction Force
    ...
    Variable "Force"
    Output Label "Top_Reaction_Force"
```

Than `Force "true"` must be defined in the `Output` section:

```
Output
  Output File Type "ExodusII"
  ..
  Output Variables
    Block_Id "false"
    ...
    Force "true"
    Top_Reaction_Force "true"
```

Without `Force "true"` an error is thrown as the group variables cannot be extracted from the global solution.

3.8.2.4. Exemplary input section

XML-format

from `test/regression/Contact_Cubes/Contact_Cubes.xml`:

```
<ParameterList name="Compute Class Parameters">
  <ParameterList name="Left Block Stored Elastic Energy">
    <Parameter name="Compute Class" type="string" value="Block_Data"/>
    <Parameter name="Calculation Type" type="string" value="Sum"/>
    <Parameter name="Block" type="string" value="block_1"/>
    <Parameter name="Variable" type="string" value="Stored_Elastic_Energy
   "/>
    <Parameter name="Output Label" type="string" value="
   Block_1_Stored_Elastic_Energy"/>
  </ParameterList>
  <ParameterList name="Right Block Stored Elastic Energy">
    <Parameter name="Compute Class" type="string" value="Block_Data"/>
    <Parameter name="Calculation Type" type="string" value="Sum"/>
    <Parameter name="Block" type="string" value="block_2"/>
    <Parameter name="Variable" type="string" value="Stored_Elastic_Energy
   "/>
    <Parameter name="Output Label" type="string" value="
   Block_2_Stored_Elastic_Energy"/>
  </ParameterList>
</ParameterList>
```

Free format

from `examples/tensile_test/tensile_test.peridigm`:

```
Compute Class Parameters
  Top Reaction Force
    Compute Class "Block_Data"
    Calculation Type "Sum"
    Block "block_3"
    Variable "Force"
    Output Label "Top_Reaction_Force"
  Bottom Reaction Force
    Compute Class "Block_Data"
    Calculation Type "Sum"
    Block "block_1"
    Variable "Force"
    Output Label "Bottom_Reaction_Force"
```

YAML format

-

3.8.2.5. List of examples

- From `examples/`:
  - `examples/tensile_test/tensile_test.peridigm`
- From `test/regression/`:
  - `Contact_Cubes/Contact_Cubes.xml`
  - `Contact_Ring/Contact_Ring.xml`
  - `WaveInBar_MultiBlock/WaveInBar_MultiBlock.xml`
- From `test/verification/`:
  - `ElasticCorrespondenceFullyPrescribedTension/ElasticCorrespondenceFullyPrescribedT`
  - `ElasticCorrespondenceQSTension/ElasticCorrespondenceQSTension.xml`
  - `ElasticPlasticCorrespondenceFullyPrescribedTension/ElasticPlasticCorrespondenceFu`
  - `IsotropicHardeningPlasticFullyPrescribedTension_NoFlaw/IsotropicHardeningPlasticF NoFlaw.xml`
  - `IsotropicHardeningPlasticFullyPrescribedTension_WithFlaw/IsotropicHardeningPlasti WithFlaw.xml`
  - `LinearLPSBar/LinearLPSBar.peridigm`
  - `ViscoplasticNeedlemanFullyPrescribedTension_NoFlaw/ViscoplasticNeedlemanFullyPres NoFlaw.peridigm`

3.8.3. Compute Class Parameters for nearest neighbor points

3.8.3.1. Description

Define user defined calculation data for the nearest point to a given spatial coordinate. These quantities can later be output, see section 3.11.5.

3.8.3.2. Code

- `/src/compute/Peridigm_Compute_Nearest_Point_Data.cpp`
- `/src/compute/Peridigm_Compute_Nearest_Point_Data.hpp`

3.8.3.3. Input parameters

List

Name	Type	Required	Default	Description
Compute Class	string	✓		"Nearest_Point_Data"
Calculation Type	string	✓	-	"Minimum" | "Maximum" | "Sum"
Block	string	✓	-	Block name
Variable	string	✓[1,3]	-	Output variable
Output Label	string	✓[2]	-	Output label for output section

Remarks

1. The output value identifier are defined in `src/io/mesh_output/Field.h`. Possible values are defined in subsubsection 3.11.1.1.
2. For the output to have any effect, the output label must be defined in the "Output" section of the input deck.
3. The item requested as `Variable` must be defined and requested as a global `Output Variable` in the `Output` block, e.g.: In case you request a `Compute Class Parameter` *Top Reaction Force* with *Variable "Force"*:

```
Compute Class Parameters
  Top Reaction Force
    ...
    Variable "Force"
    Output Label "Top_Reaction_Force"
```

Than `Force "true"` must be defined in the `Output` section:

```
Output
  Output File Type "ExodusII"
  ..
  Output Variables
    Block_Id "false"
    ...
    Force "true"
    Top_Reaction_Force "true"
```

Without `Force "true"` an error is thrown as the group variables cannot be extracted from the global solution.

3.8.3.4. Exemplary input section

XML-format

from `test/verification/NeighborhoodVolume/NeighborhoodVolume.xml`:

```
<ParameterList name="Compute Class Parameters">
  <ParameterList name="Horizon At Point A">
    <Parameter name="Compute Class" type="string" value="
   Nearest_Point_Data"/>
    <Parameter name="X" type="double" value="0.118"/>
    <Parameter name="Y" type="double" value="0.0"/>
    <Parameter name="Z" type="double" value="0.0"/>
    <Parameter name="Variable" type="string" value="Horizon"/>
    <Parameter name="Output Label" type="string" value="Horizon_Point_A
   "/>
  </ParameterList>
  <ParameterList name="Neighborhood Volume At Point A">
    <Parameter name="Compute Class" type="string" value="
   Nearest_Point_Data"/>
    <Parameter name="X" type="double" value="0.118"/>
    <Parameter name="Y" type="double" value="0.0"/>
    <Parameter name="Z" type="double" value="0.0"/>
    <Parameter name="Variable" type="string" value="Neighborhood_Volume
   "/>
    <Parameter name="Output Label" type="string" value="
   Neighborhood_Volume_Point_A"/>
  </ParameterList>
</ParameterList>
```

Free format

from `examples/tensile_test/tensile_test.peridigm`:

```
Compute Class Parameters
  Strain Gage Top Initial Position
    Compute Class "Nearest_Point_Data"
    X 0.0317
    Y 1.238
    Z 0.0
    Variable "Model_Coordinates"
    Output Label "Gage_Top_Initial_Position"
    Verbose "True"
  Strain Gage Top Displacement
    Compute Class "Nearest_Point_Data"
    X 0.0317
    Y 1.238
    Z 0.0
    Variable "Displacement"
    Output Label "Gage_Top_Displacement"
    Verbose "True"
```

YAML format

-

3.8.3.5. List of examples

- From `examples/`:
  - `examples/tensile_test/tensile_test.peridigm`
- From `test/verification/`:
  - `NeighborhoodVolume/NeighborhoodVolume.xml`
  - `SurfaceFactorCube/SurfaceFactorCube.xml`

3.9. Solver

3.9.1. Preliminaries

Peridigm supports various solver which basically represent different time integration schemes. The solver definition consists of two steps:

1. Definition of initial and final time
2. Specification of the time integration scheme

The specification of the time integration scheme is a sublist of parameters in the solver definition.

3.9.2. Explicit - Verlet

3.9.2.1. Description

Solver with explicit time integration.

Velocity-Verlet (leapfrog), keyword `Verlet`, is a time integration scheme for explicit dynamics [32]. *Peridigm* will automatically estimate a safe stable timestep for your problem. The Safety Factor multiplies this estimate. You can override the max stable timestep computed by Peridigm by specifying a timestep manually.

3.9.2.2. Code

- `/src/core/Peridigm.cpp`
  - → `PeridigmNS::Peridigm::execute`
  - → `PeridigmNS::Peridigm::executeExplicit`
- `/src/core/Peridigm.hpp`

3.9.2.3. Input parameters

List

Name	Type	Required	Default	Description
Solver parameters				
Initial Time	double	-	0.0	Start time of simulation
Final Time	double	✓	-	End time of simulation
Verbose	bool	-	false	
Time integration				
Fixed dt[1]	double	-	-	Override max stable timestep computed by *Peridigm* by specifying a timestep manually
Safety Factor[1]	double	-	1.0	Multiply stable timestep estimate returned by *Peridigm* by the user-supplied safety factor
Numerical Damping[2]	double	-	0.0	adds a numerical damping to the time integration to stabilize the dynamics caused by cracks

[2]Implemented by DLR

Remarks

1. *Fixed dt* and *Safety Factor* are mutually exclusive.
2. Specification of a timestep larger than the stable time step or a safety factor greater than one may cause instable solutions.

3.9.2.4. Exemplary input section

XML-format

```
<ParameterList name="Solver">
  <Parameter name="Verbose" type="bool" value="false"/>
  <Parameter name="Initial Time" type="double" value="0.0"/>
  <Parameter name="Final Time" type="double" value="1.0"/>
  <ParameterList name="Verlet">
    <Parameter name="Safety Factor" type="double" value="0.8"/>
    <Parameter name="Numerical Damping" type="double" value="0.02"/>
  </ParameterList>
</ParameterList>
```

Free format

```
Solver
  Verbose false
  Initial Time 0.0
  Final Time 1.0
  Verlet
    Safety Factor 0.8
    Numerical Damping 0.02
```

YAML format

-

3.9.2.5. List of examples

- From `examples/`:
  - `fragmenting_cylinder/fragmenting_cylinder.peridigm`
- From `test/regression/`:
  - `Contact_Cubes/Contact_Cubes.xml`

- From `test/verification/`:
  - `NeighborhoodVolume/NeighborhoodVolume.xml`
  - `IsotropicHardeningPlasticFullyPrescribedTension_NoFlaw/IsotropicHardeningPlasticNoFlaw.xml`

3.9.3. Implicit

3.9.3.1. Description

Solver with implicit time integration.

Implicit, keyword `Implicit`, is a time integration scheme for implicit dynamics using Newmark-beta method [32]. It uses a fixed timestep.

3.9.3.2. Code

- `/src/core/Peridigm.cpp`
  - → `PeridigmNS::Peridigm::execute`
  - → `PeridigmNS::Peridigm::executeImplicit`
- `/src/core/Peridigm.hpp`

3.9.3.3. Input parameters

List

Name	Type	Required	Default	Description
Solver parameters				
Initial Time	double	-	0.0	Start time of simulation
Final Time	double	✓	-	End time of simulation
Verbose	bool	-	false	
Time integration				
Fixed dt	double	✓	-	Fixed timestep
Absolute Tolerance	double	-	1.0E-6	
Maximum Solver Iterations	int	-	10	
Gamma	double	-	0.5	Newmark-beta parameter γ
Beta	double	-	0.25	Newmark-beta parameter β

Remarks

1. Newmark-beta-parameter combinations:
   - $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$ constant average acceleration method
   - $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{6}$ linear acceleration method

3.9.3.4. Exemplary input section

XML-format

```
<ParameterList name="Solver">
  <Parameter name="Verbose" type="bool" value="false"/>
  <Parameter name="Initial Time" type="double" value="0.0"/>
  <Parameter name="Final Time" type="double" value="0.00005"/>
  <ParameterList name="Implicit">
    <Parameter name="Fixed dt" type="double" value="0.00001"/>
    <Parameter name="Beta" type="double" value="0.25"/>
    <Parameter name="Gamma" type="double" value="0.50"/>
    <Parameter name="Absolute Tolerance" type="double" value="1.0e-10"/>
    <Parameter name="Maximum Solver Iterations" type="int" value="10"/>
  </ParameterList>
</ParameterList>
```

Free format

```
  Solver
    Verbose false
    Initial Time 0.0
    Final Time 0.00005
    Implicit
      Fixed dt 0.00001
      Beta 0.25
      Gamma 0.50
      Absolute Tolerance 1.0e-10
      Maximum Solver Iterations 10
```

YAML format

-

3.9.3.5. List of examples

- From `test/regression/`:
  - `Body_Force/Body_Force_Implicit.xml`
- From `test/verification/`:
  - `CompressionImplicit_2x1x1/CompressionImplicit_2x1x1.xml`

- `CompressionImplicitEssentialBC_2x1x1/CompressionImplicitEssentialBC_-2x1x1.xml`

3.9.4. QuasiStatic

3.9.4.1. Description

Solver with implicit time integration.

Quasi-static, keyword `QuasiStatic`, is a time integration scheme for quasi-static analysis using a nonlinear solver with modified Newton approach [32].

3.9.4.2. Code

- `/src/core/Peridigm.cpp`
  - → `PeridigmNS::Peridigm::execute`
  - → `PeridigmNS::Peridigm::executeQuasiStatic`
- `/src/core/Peridigm.hpp`

3.9.4.3. Input parameters

List

Name	Type	Required	Default	Description
Solver parameters				
Initial Time	double	-	0.0	Start time of simulation
Final Time	double	-	1.0	End time of simulation
Verbose	bool	-	false	
Disable Heuristics	bool	-	false	Disable solver heuristics
Time integration				
Number of Load Steps	int	(✓)	-	
Maximum Solver Iterations	int	-	10	
Time Steps	int	(✓)	-	
Damped Newton Diagonal Scale Factor	double	-	1.0001	
Damped Newton Diagonal Shift Factor	double	-	0.00001	
Relative Tolerance	double	-	1.0E-6	
Absolute Tolerance	double	-	✓[3]	Convergence criteria for the residual.

Remarks

1. Since the analysis is quasi-static, the values of *Initial Time* and *Final Time* are irrelevant
2. Two types of load step input are mutually exclusive, but one of them is required in the definition
   - *Final Time* and *Number of Load Steps*
   - *Time Steps*
3. A default value is calculated. In case of convergence problems, increase the value of `Absolute Tolerance`.

3.9.4.4. Exemplary input section

XML-format

```
<ParameterList name="Solver">
  <Parameter name="Verbose" type="bool" value="false"/>
  <Parameter name="Initial Time" type="double" value="0.0"/>
  <Parameter name="Final Time" type="double" value="1.0"/>
  <ParameterList name="QuasiStatic">
    <Parameter name="Number of Load Steps" type="int" value="1"/>
    <Parameter name="Absolute Tolerance" type="double" value="1.0"/>
    <Parameter name="Maximum Solver Iterations" type="int" value="10"/>
  </ParameterList>
</ParameterList>
```

Free format

```
  Solver
    Verbose "false"
    Initial Time 0.0
    Final Time 1.0
    QuasiStatic
      Number of Load Steps 1
      Absolute Tolerance 1.0
      Maximum Solver Iterations 10
```

YAML format

-

3.9.4.5. List of examples

- From `examples/`:
  - `examples/tensile_test/tensile_test.peridigm`
- From `test/regression/`:
  - `Interfaces/Interfaces.xml`
  - `Pals_Simple_Shear/Pals_Simple_Shear.xml`
  - `PrecrackedPlate/PrecrackedPlate.xml`
- From `test/verification/`:
  - `MultipleHorizons/MultipleHorizons.xml`

3.9.5. NOXQuasiStatic (nonlinear)

3.9.5.1. Description

NOX Quasi-static, keyword `NOXQuasiStatic`, is a time integration scheme for quasi-static analysis using a nonlinear solver with modified Newton approach using the NOX solver from *Trilinos*. NOX is short for Nonlinear Object-Oriented Solutions, and its objective is to enable the robust and efficient solution of the equation: $F(x) = 0$, where $F : \Re^n \rightarrow \Re^n$. NOX implements a variety of Newton-based globalization techniques including Line Search and Trust Region algorithms. In addition it provides higher and lower order models using Broyden and Tensor algorithms. Special algorithms have been developed for use with inexact linear solvers such as Krylov subspace techniques. NOX is designed to work with any linear algebra package and to be easily customized.

- Capabilities:
  - Newton-Based Nonlinear Solver
    - Linked to Trilinos linear solvers for scalability
    - Matrix-Free option
  - Anderson Acceleration for Fixed-Point iterations
  - Globalizations for improved robustness
  - Line Searches, Trust Region, Homotopy methods
  - Customizable: C++ abstractions at every level
  - Extended by LOCA package (Parameter continuation, Stability analysis, Bifurcation tracking)
- Download: Part of Trilinos
- Further information: Andy Salinger agsalin@sandia.gov

3.9.5.2. Code

- `/src/core/Peridigm.cpp`
  → `PeridigmNS::Peridigm::execute`
  → `PeridigmNS::Peridigm::executeNOXQuasiStatic`
- `/src/core/Peridigm.hpp`

3.9.5.3. Input parameters

Preconditioner keywords

Name	Type	Default	Description
Peridigm Preconditioner	String		"None", "Full Tangent"

NOXQuasiStatic keywords

Inside the group NOXQuasiStatic:

Name	Type	Default	Description
Nonlinear Solver	String		"Line Search Based"
Number of Load Steps	int		
Max Solver Iterations	int		
Relative Tolerance	double		
Max Age Of Prec	int		

Groups:

- Direction
- Line Search

Direction keywords

Inside the group Direction:

Name	Type	Default	Description
Method	String		"Newton"
Linear Solver			
Jacobian Operator	String		"Matrix-Free"
Preconditioner	String		"None", "AztecOO", "User Defined"

Line search keywords

Inside the group Line search:

Name	Type	Default	Description
Method	String		"Polynomial"

3.9.5.4. Exemplary input section

XML-format

-

Free format

```
Solver
  Initial Time 0.0
  Final Time 1.0
  Peridigm Preconditioner "Full Tangent"
  NOXQuasiStatic
    Nonlinear Solver "Line Search Based"
    Number of Load Steps 1
    Max Solver Iterations 100
    Relative Tolerance 1.0e-9
    Max Age Of Prec 100
    Direction
      Method "Newton"
      Newton
        Linear Solver
          Jacobian Operator "Matrix-Free"
          Preconditioner "AztecOO"
    Line Search
      Method "Polynomial"
```

YAML format

from `NOX_QS_MatrixFree_NoPrec_YAML.yaml`

```
Solver:
  Initial Time: 0.0
  Final Time: 1.0
  Peridigm Preconditioner: "None"
  NOXQuasiStatic:
    Nonlinear Solver: "Line Search Based"
    Number of Load Steps: 1
    Max Solver Iterations: 100
    Relative Tolerance: 1.0e-9
    Max Age Of Prec: 100
    Direction:
      Method: "Newton"
      Newton:
        Linear Solver:
          Jacobian Operator: "Matrix-Free"
          Preconditioner: "None"
    Line Search:
      Method: "Polynomial"
```

3.9.5.5. List of examples

- From `test/regression/`:
  - `Compression_NLCGQS_3x2x2/Compression_NLCGQS_3x2x2.xml`
  - `NOX_QS/NOX_QS_MatrixFree_3x3Prec.peridigm`
  - `NOX_QS/NOX_QS_MatrixFree_FullTangentPrec.peridigm`
  - `NOX_QS/NOX_QS_MatrixFree_NoPrec.peridigm`
  - `NOX_QS/NOX_QS_Newton_AztecOOPrec.peridigm`
  - `NOX_QS/NOX_QS_Newton_NoPrec.peridigm`
  - `NOX_QS/NOX_QS_MatrixFree_NoPrec_YAML.yaml`

3.9.6. Sequence of solvers

You can use multiple solvers in a sequence. These must not be identical in type of integration scheme. The sequence is defined by the keyword `Solver` plus a counter id, e.g. `Solver1`, `Solver2`, ...

Exemplary input section

XML-format

```
<ParameterList name="Solver1">
  <Parameter name="Verbose" type="bool" value="false"/>
  <Parameter name="Initial Time" type="double" value="0.0"/>
  <Parameter name="Final Time" type="double" value="0.00005"/>
  <ParameterList name="QuasiStatic">
    <Parameter name="Number of Load Steps" type="int" value="20"/>
    <Parameter name="Absolute Tolerance" type="double" value="1.0e-2"/>
    <Parameter name="Maximum Solver Iterations" type="int" value="10"/>
  </ParameterList>
</ParameterList>
<ParameterList name="Solver2">
  <Parameter name="Verbose" type="bool" value="false"/>
  <Parameter name="Initial Time" type="double" value="0.00005"/>
  <Parameter name="Final Time" type="double" value="0.0001"/>
  <ParameterList name="Verlet">
    <Parameter name="Fixed dt" type="double" value="1.0e-7"/>
  </ParameterList>
</ParameterList>
```

Free format

```
Solver1
  Verbose "false"
  Initial Time 0.0
  Final Time {TWIST_FINAL_TIME}
  QuasiStatic
    Number of Load Steps {NUMBER_OF_QS_STEPS}
```

```
    Absolute Tolerance 1.0
    Maximum Solver Iterations 100
Solver2
  Verbose "false"
  Initial Time {TWIST_FINAL_TIME}
  Final Time {PULL_FINAL_TIME}
  Verlet
    Safety Factor 0.8
```

YAML format

List of examples

- From `examples/`:
  - `twist_and_pull/twist_and_pull.peridigm`
- From `test/regression/`:
  - `Compression_QS_Explicit_3x2x2/Compression_QS_Explicit_3x2x2.xml`
  - `Compression_QS_Explicit_MultiFreqOutput_3x2x2/Compression_QS_Explicit_-MultiFreqOutput_3x2x2.xml`

3.10. Solver Tools

3.10.1. Restart

3.10.1.1. Description

Writes the last state of an analysis for use in a restart analysis.

3.10.1.2. Literature

-

3.10.1.3. Code

Release version

Available from the current master.

Required compiler options

-

Routines

- from `src/core/`:
  - `Peridigm.cpp/hpp`
  - `Peridigm_State.cpp/hpp`

3.10.1.4. Input parameters

List

Name	Type	Required	Default	Description
Restart	bool	✓[1]	-	

Remarks

1. The first and the restart model must be located in the same directory

3.10.1.5. Exemplary input section

XML-format

from `test/regression/Contact_Perforation_With_Restart/`:

Initial run:

```
<ParameterList name="Restart">
  <Parameter name="Restart" type="bool" value="true"/>
</ParameterList>

<ParameterList name="Solver">
  <Parameter name="Verbose" type="bool" value="false"/>
  <Parameter name="Initial Time" type="double" value="0.0"/>
  <Parameter name="Final Time" type="double" value="3.5e-2"/>
  <ParameterList name="Verlet">
    <Parameter name="Fixed dt" type="double" value="3.5e-5"/>
  </ParameterList>
</ParameterList>
```

Restart run:

```
<ParameterList name="Restart">
  <Parameter name="Restart" type="bool" value="true"/>
</ParameterList>

<ParameterList name="Solver">
  <Parameter name="Verbose" type="bool" value="false"/>
  <Parameter name="Initial Time" type="double" value="3.5e-2"/>
  <Parameter name="Final Time" type="double" value="7.0e-2"/>
  <ParameterList name="Verlet">
    <Parameter name="Fixed dt" type="double" value="3.5e-5"/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

3.10.1.6. List of examples

- From `test/regression/`:
  - `Contact_Perforation_Run1.xml`
  - `Contact_Perforation_Run2.xml`

3.11. Output

3.11.1. Preliminaries

Basically, there are five types of distinguished output data in *Peridigm*:

- Global scalar fieldspecs — scalar fields defined over entire simulation
- Multiphysics fieldspecs
- Element scalar fieldspecs — scalar fields defined over elements
- Nodal Vector3D fieldspecs — vector fields defined at nodes
- Bond scalar fieldspecs

All identifiers for possible output values are defined in `.\src\io\mesh_output\Field.h`.

3.11.1.1. Output variables

Global scalar

- Global_Angular_Momentum
- Global_Kinetic_Energy
- Global_Linear_Momentum
- Global_Strain_Energy
- Global_Strain_Energy_Density

Multiphysics

Not considered here.

- Fluid_Pressure_U
- Fluid_Pressure_V
- Fluid_Pressure_Y
- Flux
- Flux_Density

Element scalar

- BC_MASK
- Block_Id
- Critical_Stretch
- Critical_Time_Step
- Damage
- Density
- Dilatation
- Element_Id
- Interface_Proximity
- Kinetic_Energy
- Lambda
- Neighborhood_Volume
- Norm_td
- Num_Neighbors
- Number_Of_Neighbors
- Proc_Num
- Radius
- Strain_Energy
- Strain_Energy_Density
- Volume
- Weighted_Volume

Nodal Vector3D

- Acceleration
- Angular_Momentum
- Contact_Force
- Contact_Force_Density
- Coordinates
- Displacement
- Force
- Force_Density
- Linear_Momentum
- Model_Coordinates
- Residual
- Tangent_Reference_Coordinates
- Velocity

Bond scalar

- Bond_Damage
- Deviatoric_Plastic_Extension
- Deviatoric_Back_Extension
- Partial_Volume

The different entities are written to different output files:

- .e: Element scalar, Nodal Vector3D, Bond scalar
- .h: Global scalar

It is not differentiated between cellular data and point data. Also it is not important if the calculation supports the output. If the simulation has no results for a specific output the data will be set to zero. However, the size of the output file itself is influenced. All variables can be found with the defined names in *Paraview*.

Additional output values from classes

The following classes have their own output values. The specific output variables are stated in the class description in this document.

- Materials
- Compute classes in `src/compute/`

Compute class output values

- Stored_Elastic_Energy → Peridigm_Compute_Stored_Elastic_Energy.cpp
- Stored_Elastic_Energy_Density → Peridigm_Compute_Stored_Elastic_Energy_Density.cpp

3.11.1.2. Output data acquisition

Output data acquisition basics

The output data acquisition block starts with information about how and where to store the entities. The following entries are not mandatory:

- Output Format : default: binary
- Parallel Write : default: true

Output data acquisition without global scalar entities

In case no global scalar value is required, it is sufficient to create one output in the input deck. OUTPUT PARAMETER LIST can contain any of the aforementioned entities for the .e-results file. By default the output is set to false for all entities. The result file will be `FILENAME.e`.

Free-format

```
Output
  Output File Type ExodusII
  Output Format "BINARY"
  Output Filename "FILENAME"
  Output Frequency "1"
  Parallel Write "true"
  Output Variables
    OUTPUT PARAMETER LIST, e.g.
    Displacement "true"
    Velocity "true"
```

XML-format

```
<ParameterList name="Output">
  <Parameter name="Output File Type" type="string" value="ExodusII"/>
  <Parameter name="Output Format" type="string" value="BINARY"/>
  <Parameter name="Output Filename" type="string" value="Filename"/>
  <Parameter name="Output Frequency" type="int" value="1"/>
```

```
  <Parameter name="Parallel Write" type="bool" value="true"/>
  <ParameterList name="Output Variables">
    OUTPUT PARAMETER LIST, e.g.
    <Parameter name="Displacement" type="bool" value="true"/>
    <Parameter name="Velocity" type="bool" value="true"/>
  </ParameterList>
</ParameterList>
```

Output data acquisition including global scalar entities

Two different sets of outputs have to created, one for the .e- and one for the .h-entities. The result files will be `FILENAME.e` for `Output_1` and `FILENAME.h` for `Output_2`. The output frequencies can vary between the two outputs. The non-mandatory items are left out in this example.

Free-format

```
Output1
  Output File Type ExodusII
  Output Filename "FILENAME"
  Output Frequency "10"
  Output Variables
    .e OUTPUT PARAMETER LIST, e.g.
    Displacement "true"
    Velocity "true"

Output2
  Output File Type ExodusII
  Output Filename "FILENAME"
  Output Frequency "5"
  Output Variables
    .h OUTPUT PARAMETER LIST, e.g.
    Global_Kinetic_Energy "true"
    Global_Linear_Momentum "true"
```

XML-format

equivalent

3.11.1.3. Output sequence

A sequencially different output, e.g. for different solver steps, can be obtained via the specification of *Initial Output Step* and *Final Output Step* for each *Output* block. For the first output block *Initial Output Step* and for the last output block *Final Output Step* do not have to be specified if these coincide with 0.0 and the end of the simulation.

Exemplary input section

XML-format

-

Free format

```
Output1
  Output File Type "ExodusII"
  Output Format "BINARY"
  Output Filename "model_implicit"
  Output Frequency 1
  Final Output Step 10
  Output Variables
    Block_Id "true"
    ...
    Displacement "true"

Output2
  Output File Type "ExodusII"
  Output Filename "model_explicit"
  Output Frequency 50
  Initial Output Step 11
  Output Variables
    Block_Id "true"
    ...
    Displacement "true"
```

YAML format

-

List of examples

- From `examples/`:
  - `twist_and_pull/twist_and_pull.peridigm`

3.11.1.4. User-defined output entities

So called `Compute Class Parameters` can be defined for the acquisition of result data for nodesets, blocks or the nearest point next to a specific location.

The respective keywords are explained in sections 3.8.1, 3.8.2 and 3.8.3

3.11.2. Output

3.11.2.1. Code

- src/core/Peridigm.cpp
- src/io/Peridigm_OutputManager_ExodusII.cpp

3.11.2.2. Input parameters

List

Name	Type	Required	Default	Description
Base parameters				
Output File Type	string	-	"ExodusII"	"ExodusII"
Output Format	string	-	"BINARY"	"BINARY" | "ASCII"
Output Filename[1]	string	-	"dump"	Output file name
Output Frequency	int	✓	-	
Initial Output Step	int	-	1	First time step this output shall apply to
Final Output Step	int	-	last in model	Last time step this output shall apply to
Parallel Write	bool	-	true	Flag whether to write in parallel
Output Variables[2]	list	✓	-	Opens list of output variables

Remarks

1. The *Output Filename* should be unique for each *Output*. Otherwise, output files are probably overwritten.
2. See section 3.11.1.1

3.11.2.3. Exemplary input section

XML-format

from `Body_Force_QS.xml`:

```
<ParameterList name="Output">
  <Parameter name="Output File Type" type="string" value="ExodusII"/>
  <Parameter name="Output Format" type="string" value="BINARY"/>
  <Parameter name="Output Filename" type="string" value="Body_Force_QS"/>
  <Parameter name="Output Frequency" type="int" value="1"/>
```

```
  <Parameter name="Parallel Write" type="bool" value="true"/>
  <ParameterList name="Output Variables">
    <Parameter name="Displacement" type="bool" value="true"/>
    <Parameter name="Velocity" type="bool" value="true"/>
    <Parameter name="Element_Id" type="bool" value="true"/>
    <Parameter name="Proc_Num" type="bool" value="true"/>
    <Parameter name="Dilatation" type="bool" value="true"/>
    <Parameter name="Force_Density" type="bool" value="true"/>
    <Parameter name="Weighted_Volume" type="bool" value="true"/>
  </ParameterList>
</ParameterList>
```

Free format

from `tensile_test.peridigm`: from `NOX_QS_MatrixFree_NoPrec_YAML.yaml`:

```
Output
  Output File Type "ExodusII"
  Output Filename "tensile_test"
  Output Frequency 1
  Output Variables
    Displacement "true"
    Velocity "true"
    Element_Id "true"
    Proc_Num "true"
    Force_Density "true"
    Hourglass_Force_Density "true"
    Force "true"
    Volume "true"
```

YAML format

from `NOX_QS_MatrixFree_NoPrec_YAML.yaml`:

```
Output:
  Output File Type: "ExodusII"
  Output Filename: "NOX_QS_MatrixFree_NoPrec_YAML"
  Output Frequency: 1
  Output Variables:
    Displacement: "true"
    Velocity: "true"
    Force: "true"
    Volume: "true"
```

```
    Radius: "true"
    Nonlinear_Solver_Iterations: "true"
```

3.11.2.4. List of examples

Basically all models in `/test/` and `/examples/`.

3.11.3. User defined results for node sets

3.11.3.1. Description

User defined output for node sets using `Compute Class Parameters` from section 3.8.1.

3.11.3.2. Exemplary input section

Requirement is the definition of the respective Compute Class Parameters from section 3.8.1.

XML-format

from `test/regression/Contact_Ring/Contact_Ring.xml`:

```
<ParameterList name="Output Data 1">
  <Parameter name="Output File Type" type="string" value="ExodusII"/>
  <Parameter name="Output Format" type="string" value="BINARY"/>
  <Parameter name="Output Filename" type="string" value="Contact_Ring"/>
  <Parameter name="Output Frequency" type="int" value="10"/>
  <Parameter name="Parallel Write" type="bool" value="true"/>
  <ParameterList name="Output Variables">
    ...
    <Parameter name="Stored_Elastic_Energy" type="bool" value="true"/>
    <Parameter name="Global_Kinetic_Energy" type="bool" value="true"/>
    ...
    <Parameter name="NS_10_Stored_Elastic_Energy" type="bool" value="true
   "/>
    <Parameter name="NS_11_Stored_Elastic_Energy" type="bool" value="true
   "/>
  </ParameterList>
</ParameterList>
```

Free format

-

YAML format

-

3.11.3.3. List of examples

- From `test/regression/`:
  - `Contact_Ring/Contact_Ring.xml`
  - `WaveInBar_MultiBlock/WaveInBar_MultiBlock.xml`

3.11.4. User defined results for blocks

3.11.4.1. Description

User defined output for blocks using `Compute Class Parameters` from section 3.8.2.

3.11.4.2. Exemplary input section

Requirement is the definition of the respective Compute Class Parameters from section 3.8.2.

XML-format

from `test/regression/Contact_Cubes/Contact_Cubes.xml`:

```
<ParameterList name="Output Data">
  <Parameter name="Output File Type" type="string" value="ExodusII"/>
  <Parameter name="Output Format" type="string" value="BINARY"/>
  <Parameter name="Output Filename" type="string" value="Contact_Cubes"/>
  <Parameter name="Output Frequency" type="int" value="1000"/>
  <Parameter name="Parallel Write" type="bool" value="true"/>
  <ParameterList name="Output Variables">
    ...
    <Parameter name="Stored_Elastic_Energy" type="bool" value="true"/>
    <Parameter name="Global_Kinetic_Energy" type="bool" value="true"/>
    ...
    <Parameter name="Block_1_Stored_Elastic_Energy" type="bool" value="
   true"/>
    <Parameter name="Block_2_Stored_Elastic_Energy" type="bool" value="
   true"/>
  </ParameterList>
</ParameterList>
```

Free format

from `examples/tensile_test/tensile_test.peridigm`:

```
Output
  Output File Type "ExodusII"
  Output Filename "tensile_test"
  Output Frequency 1
  Output Variables
```

```
...
Force "true"
...
Top_Reaction_Force "true"
Bottom_Reaction_Force "true"
...
```

YAML format

-

3.11.4.3. List of examples

- From `examples/`:
  - `examples/tensile_test/tensile_test.peridigm`
- From `test/regression/`:
  - `Contact_Cubes/Contact_Cubes.xml`
  - `Contact_Ring/Contact_Ring.xml`
  - `WaveInBar_MultiBlock/WaveInBar_MultiBlock.xml`
- From `test/verification/`:
  - `ElasticCorrespondenceFullyPrescribedTension/ElasticCorrespondenceFullyPrescribedT`
  - `ElasticCorrespondenceQSTension/ElasticCorrespondenceQSTension.xml`
  - `ElasticPlasticCorrespondenceFullyPrescribedTension/ElasticPlasticCorrespondenceFu`
  - `IsotropicHardeningPlasticFullyPrescribedTension_NoFlaw/IsotropicHardeningPlasticF NoFlaw.xml`
  - `IsotropicHardeningPlasticFullyPrescribedTension_WithFlaw/IsotropicHardeningPlasti WithFlaw.xml`
  - `LinearLPSBar/LinearLPSBar.peridigm`
  - `ViscoplasticNeedlemanFullyPrescribedTension_NoFlaw/ViscoplasticNeedlemanFullyPres NoFlaw.peridigm`

3.11.5. User defined results for nearest neighbor points

3.11.5.1. Description

User defined output for the nearest point to a given spatial coordinate using `Compute Class Parameters`.

3.11.5.2. Exemplary input section

Requirement is the definition of the respective Compute Class Parameters from section 3.8.3.

XML-format

from `test/verification/NeighborhoodVolume/NeighborhoodVolume.xml`:

```
<ParameterList name="Output">
  <Parameter name="Output File Type" type="string" value="ExodusII"/>
  <Parameter name="Output Filename" type="string" value="
   NeighborhoodVolume"/>
  <Parameter name="Output Frequency" type="int" value="1"/>
  <ParameterList name="Output Variables">
    ...
    <Parameter name="Horizon" type="bool" value="true"/>
    <Parameter name="Neighborhood_Volume" type="bool" value="true"/>
    ...
    <Parameter name="Horizon_Point_A" type="bool" value="true"/>
    <Parameter name="Neighborhood_Volume_Point_A" type="bool" value="true
   "/>
    ...
  </ParameterList>
</ParameterList>
```

Free format

from `examples/tensile_test/tensile_test.peridigm`:

```
Output
  Output File Type "ExodusII"
  Output Filename "tensile_test"
  Output Frequency 1
  Output Variables
```

```
...
Displacement "true"
...
Gage_Top_Initial_Position "true"
...
Gage_Top_Displacement "true"
...
```

YAML format

-

3.11.5.3. List of examples

- From `examples/`:
  - `examples/tensile_test/tensile_test.peridigm`
- From `test/verification/`:
  - `NeighborhoodVolume/NeighborhoodVolume.xml`
  - `SurfaceFactorCube/SurfaceFactorCube.xml`

4. Run *Peridigm*

4.1. Execution

4.1.1. On a local machine or a virtual box

4.1.1.1. Input

Peridigm is run from the command line and requires both an input deck (text file) and a discretization. Input decks may be in either of the formats described in section 2.2.

Peridigm is capable of operating on several types of discretizations. The different types of discretizations are described in section 2.3.

An example command for running *Peridigm* from the command line:

```
Peridigm my_input.peridigm
```

Peridigm is meant to be run in a massively parallel setting. In order to use multiple processors in combination with *Exodus/Genesis* mesh files (*.g) parallel decomposition of these files has to performed first as a pre-processing step when running multi-processor *Peridigm* simulations. The *decomp* utility, provided by the *SEACAS Trilinos* package, provides a mechanism for decomposing an *Exodus/Genesis* mesh file. An example command for partitioning an *Exodus/Genesis* mesh file with *decomp* for four processors:

```
decomp -p 4 my_mesh.g
```

If the *decomp* tool is not available in the `PATH` variable use:

```
/usr/local/bin/trilinos-12.4.2/bin/decomp -p 4 my_mesh.g
```

Afterwards, *Peridigm* can be run in parallel with the same amount of threads with:

```
mpirun -np 4 Peridigm my_input.peridigm
```

Text file discretizations do not require this pre-processing step, they are partitioned automatically by *Peridigm*.

4.1.1.2. Output

Preliminary informations

Peridigm generates output in the *Exodus* file format. The content of an *Exodus* output file is dictated by the `Output` section of a *Peridigm* input deck. Output may include primal quantities such a nodal displacements and velocities, as well as derived quantities such as stored elastic energy. The *ParaView* visualization code is recommended for viewing *Peridigm* results. Additional options for parsing output data are available within the *SEACAS Trilinos* package.

Exodus is based upon *NetCDF-C*, and also includes the nemesis parallel extensions. *Exodus* is fundamentally a serial output format, and each MPI process will open and write its own Exodus database for all the elements that it owns. If a load rebalance occurs during the course of a parallel simulation, each MPI process must close its Exodus database and open a new database containing the elements owned by that process after the rebalance. The net effect is that many *Exodus* output files may be generated during the course of a parallel simulation. Tools in the *SEACAS* package to help in merging these separate files together into a single Exodus database.

Merge result files of MPI executions of *Peridigm*

To automate the merging of result files for MPI executions of *Peridigm*, a python script is distributed with *Peridigm*. The file is located in `../scripts/MergeFiles.py` of the *Peridigm* installation directory. This script will produce a single *Exodus* database file containing the complete simulation output.

Before you can access the script you have to add read and execution rights to the *Peridigm* folders as root:

```
chmod -R go+rx /usr/local/src/Peridigm-1.4.1*
```

To use the script, create a symbolic link in the model directory, where the individual output files in the *Exodus* format are located:

```
ln -sf /usr/local/src/Peridigm-1.4.1/scripts/MergeFiles.py
```

Afterwards, execute the script with the number of processors and the model name as parameters.

```
./MergeFiles.py <File Base Name> <# of Processors>
```

For the `fragmented_cylinder.peridigm` example from the *Peridigm* examples directory and an initial calculation using four processors, the call to the script is

```
./MergeFiles.py fragmented_cylinder 4
```

which merges the result files `fragmented_cylinder.e.0` ... `fragmented_cylinder.e.3` of each individual processor to `fragmented_cylinder.e`. This is the final results file and should be used for postprocessing in *ParaView*.

4.1.1.3. Examples

The most effective way to learn how to use *Peridigm* is to run the example problems in the *Peridigm* `/examples/` directory. These simulations were designed to highlight the most commonly-used features of *Peridigm*, including constitutive models, bond-failure rules, contact, explicit and implicit time integration, and I/O commands.

4.2. Best practices

4.2.1. Bridge "time" until failure occurs

For stability reasons it is currently benefitial to calculate the damage behavior of a structure using the explicit solver. Dependent on the material properties and loads, the model has to sustain a certain amount of displacement before initial failure occurs. The solution domain is commonly of little interest. Thus, some ways are proposed to get to the state where failure occurs faster.

4.2.1.1. Use a combination of multiple solvers

Use a sequence of a quasi-static and an explicit solver as described in subsection 3.9.6.

This way might not be possible in case the model and therefore stiffness matrix is too large for the computational hardware. In this case, the tangent stiffness matrix cannot be allocated for the quasi-static solver. Instead, the idea proposed in the next section can be applied.

4.2.1.2. Use an initial displacement

If a continuous displacement field is present before failure, e.g. bridging the gap between an impactor and the structure or a uniform tensile test, an initial displacement can be applied on the collocation point nodes of the *complete* moving structure.

It is not possible to just apply the initial displacement on the boundary condition region. If the initial displacement is larger than the horizon, the model will fall apart.

To get the correct offset

```
#{OFFSETORIGINX=-23.0/2.0}
#{FAILUREDISPLACEMENTINIT=0.900}
#{FAILUREDISPLACEMENTFINAL=1.100}
...
Boundary Conditions
  ...
  InitialDisplacement-3-D-x
    Type "Initial Displacement"
    Node Set "moving_x"
    Coordinate "x"
    Value "value={FAILUREDISPLACEMENTINIT}*(x-{OFFSETORIGINX})"
  Displacement-1-V-x
    Type "Prescribed Displacement"
    Node Set "bc_load"
```

```
    Coordinate "x"
    Value "value={FAILUREDISPLACEMENTINIT}+{VELOCITY}*t"
...
Solver
  Verbose "true"
  Final Time {(FAILUREDISPLACEMENTFINAL-FAILUREDISPLACEMENTINIT)/VELOCITY}
  Verlet
    Safety Factor 0.9
```

4.2.2. Increase timestep in explicit solver

The critical time step calculation is performed in the `PeridigmNS::ComputeCriticalTimeStep` method in the class `/src/core/Peridigm_CriticalTimeStep.cpp`. The calculation is carried out for each block as requested in `PeridigmNS::Peridigm::executeExplicit` of `src/core/Peridigm.cpp`. The minimum of all block values is taken as the final critical time step.

The equation used for the determination of the critical time step is

$$\Delta t_{\text{Block}} = \sqrt{\frac{2\rho}{\Delta t_{\text{Denominator}}}} \tag{4.1}$$

The time step denominator itself is a function of the collocation point volume, neighbor distance and "bond" stiffness. Despite implementing the state-based peridynamics, *Peridigm* uses the bond constant for 3D bond-based peridynamics from Table 2.3 as the the "bond" stiffness. Thus

$$\Delta t_{\text{Denominator}} = f\left(\Delta x, K, \delta^{-4}\right) \tag{4.2}$$

The following actions can be taken to influence the time step:

- Increase material density
  - For quasi-static simulations a certain amount is possible
  - Reduces oscillation effects due to increased mass intertia
  - Limit is aperiodic boundary case
- Increase element size:
  - A larger element size increases the volume associated with each peridynamic collocation point
  - On the other hand the initial distance between points is increased
  - Thus, do not expect a linear response of the time step for changing mesh sizes
- Increase horizon

4.2.3. Create pre-cracks

See subsection 3.2.2

I suggest you output the variable Number_Of_Neighbors in your output file, so that you can confirm that your pre-crack is in the right place. It always seems to take me multiple tries to get it right.

5. Pre- and postprocessing with *ParaView*

5.1. Input & Import

Peridigm creates result files in the *Exodus* format. This format can read by *ParaView*. It is basically the same format as the *Exodus* finite element mesh discretization. Therefore, also the underlying *Exodus* mesh can be visualized in *ParaView*.

If multiple processors are used for the execution of *Peridigm* each MPI core creates an individual *Exodus* output file. The individual result files must be merged before the output can be used in *ParaView*. For the proper procedure please consult section 4.1.1.2.

To import a file to *ParaView* perform the following steps

1. From the menu bar:
   - Click File
   - Click Open
   - Select the .g/.e-*Peridigm* input or output file

   or click  in the Main Controls toolbar
2. In the newly opened *Properties* tab:
   - Choose the variables you want to use for post-processing
   - Choose the blocks, assemblies and material regions you want to include
   - Click *Apply*

5.2. Result data

ParaView distinguishes three types of result data:

- point data
- cell data
- field data

Point data is specified at each grid point. Cell data is specified per cell. Field data occurs for user-defined output values. Several linear and non-linear cell types currently exist for one-, two- or three-dimensional applications. E.g. a cell can be a quadrilateral between four nodes in two dimensions, a hexahedron volume between eight nodes in three dimensions or others. All cell types are shown in [33].

Following is a list of *Peridigm* post-processing items and their classification as point data or cell data.

- Point data results:
  - Contact_Force
  - Displacement
  - Force
  - Global Node Ids
  - Velocity
  - ...
- Field data results:
  - Compute Class Parameter results
- Cell data results:
  - Damage
  - Dilatation
  - Element_Id
  - Global Element Ids
  - Number_Of_Neighbors
  - Object IDs
  - Proc_num
  - Radius
  - Weighted_Volume
  - ...

5.3. Selection

5.3.1. Create a selection

Selections are

- a mechanism to identify subset of a dataset
- Focus on selected subset:
  - Inspect properties of the subset
  - Further process subset alone
- Track subset over time

Selections are performed in a toolbar in the RenderView. At first it has to be chosen if the items should be added to or removed from the selection using the following buttons:

 Add selection  Subtract selection  Toggle selection

In a second step the kind of item, cells or points, to be selected has to be specified:

Points:		Select points on		Select points through
		Select points with polygon		Select points interactive
Cells:		Select cells on		Select cells through
		Select cells with polygon		Select cells interactive

Selection steps can be repeated as long as no Filter dependent on the selection is active. The currently selected entities are highlighted.

5.3.2. Limit display to selection

To limit the model to a user defined selection:

1. Import result file to *ParaView*
2. Create a selection
3. From the menu bar:
   - Click Filters
   - Click Data Analysis
   - Click *Extract Selection*

   or click  in the Data Analysis toolbar
4. Click *Apply* in the newly opened Properties tab

5.3.3. Remarks

1. Selected points lie in a virtual box. If the deformation of the model are to big, the points move outside the box and the allocation is lost. This might be an issue for plotting data. This can be solved by setting the deformation scaling to zero.

5.4. View & Display

5.4.1. Save view

To save a view configuration click an the *Adjust Camera* button  in the icon bar of your *RenderView*.

5.4.2. Display point/node numbers

After importing the model it might be of interest to get the node/point numbers for the whole model or a specific selection. To show these numbers after importing the model:

1. Generate and extract the selection of interest according to section 5.3
2. Left-click on the imported model (set the model active)
3. From the menubar:
   - Click *View*
   - Click *Selection Display Inspector*
4. In the *Selection Display Inspector* window:
   - Click on *Point Labels*
   - Select *ID*
5. The point/node numbers appear as labels

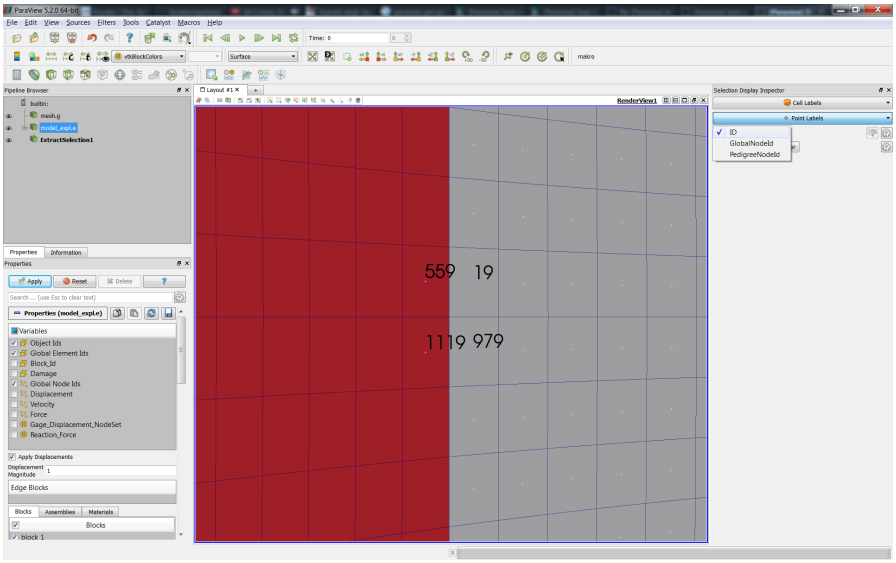


Figure 5.1.: Display of selection point/node numbers

5.4.3. Visualize nodesets

To visualize the nodesets from a *Exodus* mesh or *Peridigm* results file in *ParaView* perform the following steps

1. From the menu bar:
   - Click File
   - Click Open
   - Select the .g/.e-*Peridigm* input or output file

   or click  in the Main Controls toolbar
2. In the newly opened *Properties* tab:
   - Click the gear symbol  next to the *Search* line
   - *Sets* and *Maps* tables become available for selection
   - Select the nodesets to display
   - Unselect all Blocks
   - Click *Apply*
3. Create a *Glyph* based on the current model as described in subsection 5.4.4 (without the result data stuff of course)
4. Repeat step 1
5. In the newly opened *Properties* tab:
   - Select all Blocks
   - Set the *Opacity* to 0.4
   - Click *Apply*

Similarly, the nodesets in the peridynamic collocation point translation can be visualized. The following figure shows the base *Exodus* finite element mesh in gray. Superimposed with the finite element mesh are the nodes of a finite element nodeset (blue) and the nodeset in the peridynamic translation (green).

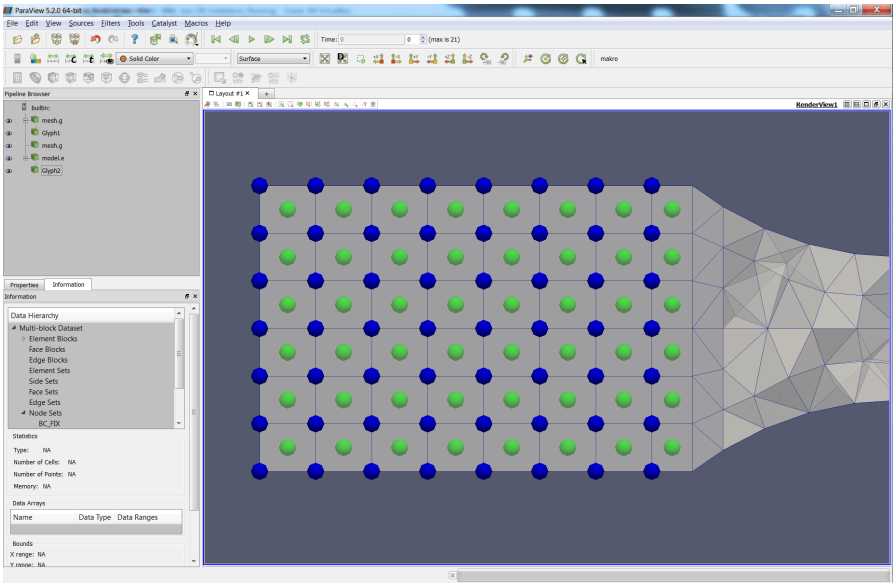


Figure 5.2.: Visualization of base finite element mesh (gray), finite element nodeset (blue) and nodeset after transformation to peridynamic collocation points (green)

5.4.4. Damage plot on nodes as spheres

Commonly, the *Peridigm* results are plotted on so called `Glyphs` in *ParaView*. A `Glyph` is a geometric object with a specific size, a direction and a color, which is drawn at certain positions within the vector field. `Glyph` shapes can be an

- Arrow
- Box
- Line
- 2D glyph
- Cone
- Cylinder
- Sphere

A natural choice for peridynamic nodes is the sphere. Unfortunately, cell data can not be plotted directly in a `Glyph` plot. Therefore, the cell data has to be converted to point data first.

1. Import result file to *ParaView*
2. Left-click on the result file once in the Pipeline Browser (mark the result file)
3. From the menu bar:
   - Click Filters
   - Click Alphabetical
   - Click *Cell Data to Point Data*
4. Left-click on the new item *Cell Data to Point Data* in the Pipeline Browser
5. From the menu bar:
   - Click Filters
   - Click Common

- Click *Glyph*

or choose the *Glyph*-symbol from the common filter icon toolbar

6. In the *Glyph* properties window choose:
   - Glyph Type: *Sphere*
   - Scalars: *Damage*
   - Vectors: *Displacement*
   - Scale factor: Choose according to your needs
   - Glyph mode: *All Points*
   - Coloring: *Damage*

   and click *Apply*
7. Afterwards, return to the properties window, go to the `Display` section and choose:
   - Representation: *Surface*
   - Coloring: *Damage*

Afterwards, you can skip through the time steps of your analysis in the Current Time Controls toolbar. It maybe necessary to adjust the range of the legend to the current time step minimum or maximum.

5.5. Plotting

5.5.1. Plot selection data over time

To plot selection data, e.g. point results, over calculation time perform the following steps:

1. Import result file to *ParaView*
2. Create a selection, e.g. to a single or two points
3. From the menu bar:
   - Click Filters
   - Click Data Analysis
   - Click *Plot Selection Over Time*

   or click  in the Data Analysis toolbar
4. In the newly opened Properties tab:
   - Review the Copied Selection
   - Click *Apply*
   - Let *ParaView* process your request a couple of seconds
   - For the actual values deselect the *Only Report Selection Statistics* checkbox
   - Click *Apply*
   - Wait until the progressbar in the bottom right is completed
   - Select from the selection entities the items you are currently interested in
   - Choose the *Series Parameters* you are interested in (there is the same quantity for each selection entity, so a couple of times if you have multiple entities in your selection, choose wisely)
   - Your requested values should show up in the *QuartileChartView*
5. You can skip through the time steps of your analysis in the Current Time Controls toolbar. The current time is shown as the vertical bar in the chart view

To see the actual values in a spreadsheet:

1. Right-click on the QuartileChartView top line (where e.g. QuartileChartView1 stands)
2. Create a selection, e.g. to a single or two points
3. Select:
   - Convert To ...
   - Spreadsheet view
4. In case you want to return to the Chart View you can do so accordingly but you have to repeat the data selection steps mentioned above

A combined view:

To save the data for plotting in `pgfplots`:

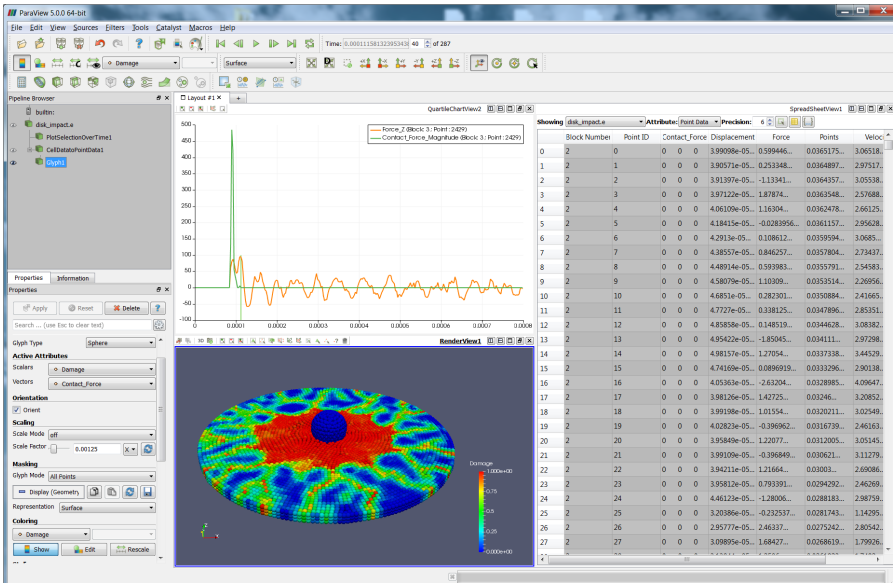


Figure 5.3.: Combined *ParaView* view

1. Select the PlotOverSelection in the *Pipeline Browser* or left click in the ChartView or SpreadsheetView
2. From the menu bar:
   - Click File
   - Click Save Data

   or click  in the Main Controls toolbar
3. Select folder and filename
4. Select csv file type
5. Click *Ok*

Beware, an individual csv-file is written for each and every entity in your selection. The file does not include any information on the entity ID or name. Thus, a sequential save-process for each single individual entity is advised. After each save, rename the resulting file.

5.5.2. Force-displacement-plots

5.5.2.1. General

This plot shows the course of the "nodal" force in a constrained region of the model over the displacement of a discrete point. The sum of the forces is used as a representative of the integral force value a load cell would show. For the displacement the value of a

discrete node or collocation point is used as a representative for the scalar value of an extensiometer or the machine displacement value.

5.5.2.2. Peridigm/ParaView

Requisition of output values in Peridigm

To acquire forces and displacements for a defined part of a model `Compute Class Parameters` are used in Peridigm. They are described in sections 3.8.1, 3.8.2 & 3.8.3. The forces are requested for the node set or block where a body load or in this case boundary condition is applied on. The displacement is written for one point in this node set using the nearest neighbor approach to a defined spatial coordinate in the node set region. The initial position of the point the displacements are written for is also requested for convenience and checks.

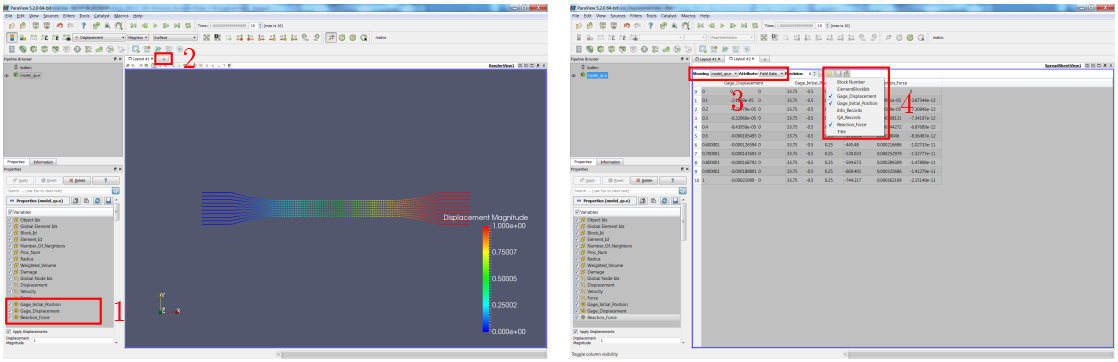
```
Compute Class Parameters
  Strain Gage Initial Position
    Compute Class "Nearest_Point_Data"
    X 0.0317
    Y 1.238
    Z 0.0
    Variable "Model_Coordinates"
    Output Label "Gage_Initial_Position"
    Verbose "True"
  Strain Gage Displacement
    Compute Class "Nearest_Point_Data"
    X 0.0317
    Y 1.238
    Z 0.0
    Variable "Displacement"
    Output Label "Gage_Displacement"
    Verbose "True"
  Reaction Force
    Compute Class "Node_Set_Data"
    Calculation Type "Sum"
    Node Set "bc_load"
    Variable "Force"
    Output Label "Reaction_Force"
```

Alternatively, it should also be possible to request the displacement for the same node set as the force and simply choose "Minimum" or "Maximum" as *Calculation type*. This way, it is not required to adjust the spatial coordinates as in *Compute Class "Nearest_Point_Data"*. This works fine for a test case.

Additionally the global output variable types *Displacement* and *Force* must be requested in the output section.

```
Output
  Output File Type "ExodusII"
  Output Filename "model"
  Output Frequency 1
  Output Variables
    ...
    Displacement "true"
    ...
    Force "true"
    ...
    Gage_Initial_Position "true"
    Gage_Displacement "true"
    Reaction_Force "true"
```

Access result data in ParaView



(a) Steps 1-2 (b) Steps 3-4

Figure 5.4.: Access `Compute Class Parameters` results data in ParaView

1. Make sure that the requested result data from the *Compute Class Parameters* is available in your results file. If you are only interested in these parameters, unselect all other.
2. Add a new view & select *SpreadSheet View*
3. For *Showing* select your model and for *Attribute* select *Field Data*
4. Click  to select the columns of interest
5. Click  to save the spreadsheet data as a `csv`-file
   a) Specify a filename & click *Ok*

b) In the next dialog toggle *Filter Columns by Visibility*. Otherwise, all data is written to the `csv`-file.

6. Do with the data whatever you want ☺

Plot result data directly in ParaView

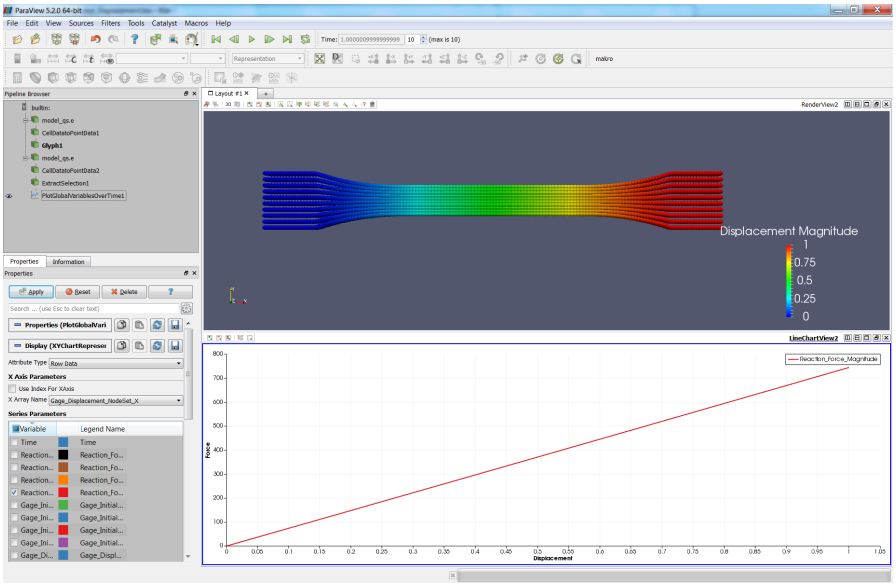


Figure 5.5.: Plot `Compute Class Parameters` results data in ParaView

1. Import your model
2. Make sure that the requested result data from the *Compute Class Parameters* is available in your results file. If you are only interested in these parameters, unselect all other.
3. From the menu bar:
   - Click Filters
   - Click Alphabetical
   - Click *Cell Data to Point Data*
4. Select points of interest:
   - Select the *Cell Data to Point Data* entry in the Pipeline Browser
   - Create a new selection of a single or multiple points in your constraint region as described in subsection 5.3.1
   - Extract the selection as described in subsection 5.3.2
5. From the menu bar:
   - Click Filters
   - Click Data Analysis

- Click *Plot Selection Over Time*

or click  in the Data Analysis toolbar

6. Preferences in the *Properties* window
   - Under *X Axis Parameters* select your displacement component or magnitude from the `Compute Class Parameters` in the *X Array Name* combobox
   - For *Series Parameters* select the force value you want to plot from the `Compute Class Parameters`, either as a component or magnitude
   - Input a *Left Axis Title* and *Bottom Axis Title* if required
7. Click *Apply*
8. In case you want to export the chart data from here:
   - Left-click on the chart
   - From the menu bar:
     - Click *File*
     - Click *Save As*
     - Export as `csv`-file

5.5.2.3. Abaqus

Standard

1. Choose Displacement (U) & Reaction Forces (RF) as field outputs for the step you want to have the plot
2. Run the analysis
3. Open the odb
4. Reaction force over time:
   a) Click *Create XY Data*
   b) Select Source *ODB history output*
   c) Click *Continue*
   d) Select all entries with beginning *Reation force:* of the nodeset your are interested in
   e) Click *Save as*
   f) Assign name, here "RF", & in *Save Operation* select *sum((XY,XY,...))*
5. Displacement over time:
   a) Click *Create XY Data*
   b) Select Source *ODB history output*
   c) Click *Continue*
   d) Select all entries with beginning *Spatial displacement Ui:* of the nodeset your are interested in
   e) Click *Save as*
   f) Assign name, here "U", & in *Save Operation* select *avg((XY,XY,...))*
6. Reaction force over displacement:

a) Click *Create XY Data*
b) Select Source *Operate on XY data*
c) Click *Continue*
d) From *Operators* select *Combine*
e) In the equation, at first select "U", than "RF", the equation than should be `combine ("U","F")`
f) Select *Plot Expression*

Explicit

1. Choose Displacement (U), Reaction Forces (RF) & Nodal forces (NFORC) as field outputs for the step you want to have the plot
2. Run the analysis
3. Open the odb
4. Reaction force & Displacement over time:
   a) Click *Create XY Data*
   b) Select Source *ODB field output*
   c) Click *Continue*
   d) *XY Data from ODB Field Output* dialog:
      i. In the *Variables* tab, select *Unique nodal* in the *Position* combobox and all necessary entry checkboxes, here RF1 and U1
      ii. In the *Elements/Nodes* tab, select the requested nodeset
   e) Click *Save*
5. Sum forces:
   a) Click *Create XY Data*
   b) Select Source *Operate on XY data*
   c) From *Operators* select *sum*
   d) In the equation, select all "RF1" entries" and click *Add to expression*
   e) Click *Save as* & assign name "RF"
6. Average displacements:
   a) Click *Create XY Data*
   b) Select Source *Operate on XY data*
   c) From *Operators* select *avg*
   d) In the equation, select all "U1" entries" and click *Add to expression*
   e) Click *Save as* & assign name "U"
7. Reaction force over displacement:
   a) Click *Create XY Data*
   b) Select Source *Operate on XY data*
   c) Click *Continue*
   d) From *Operators* select *Combine*

e) In the equation, at first select “U”, than “RF”, the equation than should be `combine ("U","F")`
f) Select *Plot Expression*

Save XY data

1. Open the odb
2. Create all results
3. In the menu bar, click *Report*
4. Click *XY*
5. Adjust the dialog preferences according to your needs

5.6. Measuring

5.6.1. Node/point position

Sometimes it is necessary to get the exact position of a single point or selection of points. The following steps give you these informations and are explained for a selection of points. For an inidividual point some steps can be omitted.

1. Import the model
2. Generate and extract the selection of interest according to section 5.3
3. Display the node/point numbers according to subsection 5.4.2
4. Remember the node/point numbers
5. In the RenderView select *Split Horizontal*
6. Create a new SpreadSheet View
7. In the SpreadSheet View:
   - Select your selection in the *Showing* combobox
   - Select *Point Data* in the *Attribute* combobox
   - The coordinates appear in the *Points* column of the table
   - In case not all your selected points appear in the table, check if the blocks the points are part of are selected in the *Properties tab* of your selection
8. Unfortunately the node/point number labels disappear

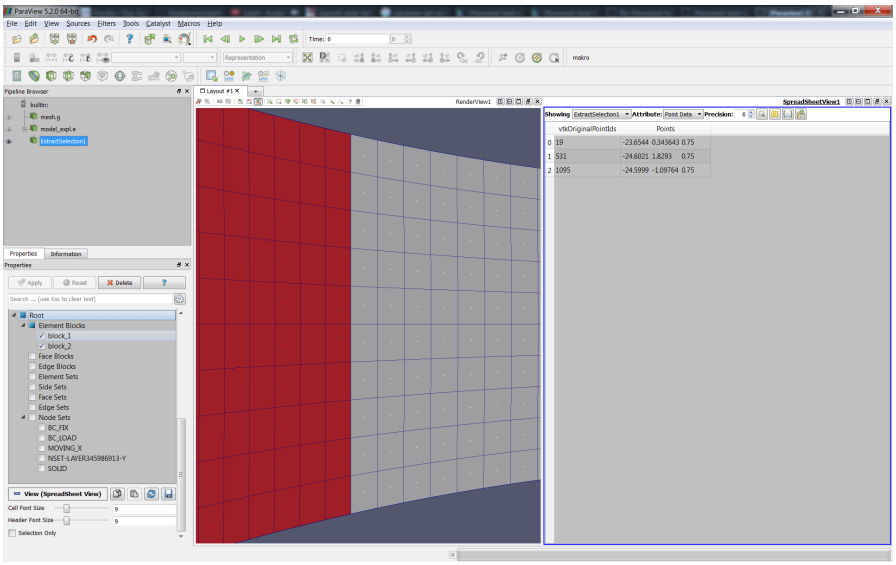


Figure 5.6.: Display of selection point/node positions

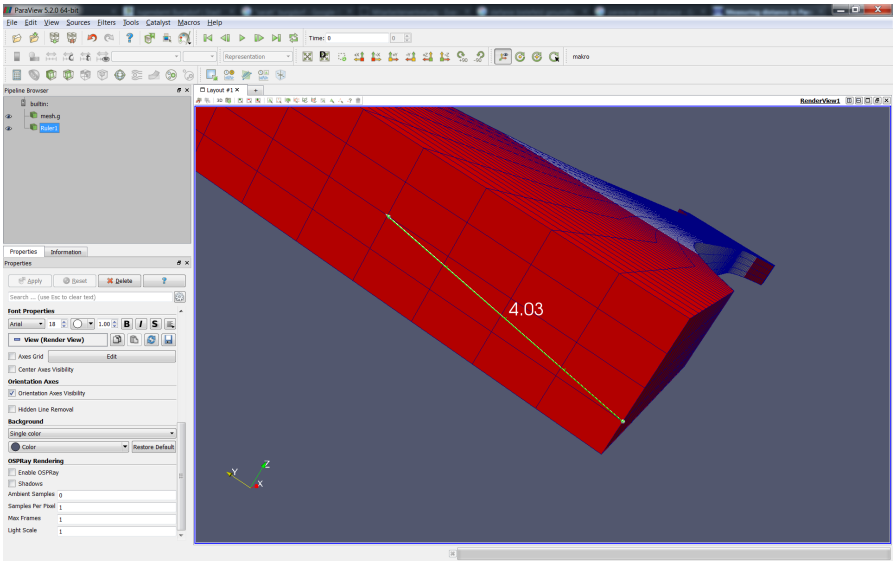


Figure 5.7.: Measure point distance in ParaView

5.6.2. Node/point distance

To measure the distance between two nodes or collocation points after importing the model:

1. In the menu bar
   - Click on *Sources*
   - Click *Ruler*
2. In the render view
   - Left-click the first point and keep the mouse clicked
   - Move the point close to the target
   - Press "CTRL+1" and snap the point to the closest mesh point
   - Do the same for the second point, but press "CTRL+2" instead for snapping
3. In the *Ruler* properties window
   - Adjust preferences to your need
   - Click *Apply*
4. The number next to the *Ruler* in the render view is the point distance

5.7. Exporting

5.7.1. Quick Screenshot

Screenshot are taken from the current view you see in *ParaView* render view. Therefore, perform all adjustments to the view to your needs before creating a screenshot.

To save a screenshot from *ParaView* perform the following steps:

1. Click *File* in the menubar
2. Click *Save Screenshot*
3. Set plot preferences:
   - Uncheck *Save only selected view* if necessary
   - Activate the *Lock aspect* button right of the resolution
   - Change the resolution to values high enough for a print-quality picture (the higher value should at least be 1000)
   - *Select image quality* slider 100
   - Override color palette: *Current palette*
   - Stereo Mode: *No stereo*
4. Click *Ok*
5. Specify the path and *File name*
6. Choose *PNG image* as file type
7. Click *Ok*

Afterwards, use *GIMP*, *Inkscape*, *convert* utility from *ImageMagick* or any other tool to convert the pixel to a non-scalable vector graphics copy as `eps` and `pdf` copy.

For the sake of reproducibility of the created picture save the *ParaView* state as described in section 5.7.4.

The target must be to have the figure and the *ParaView* state file available at all time:

```
figure_name.eps
figure_name.pdf
figure_name.png
figure_name.pvsm
```

5.7.2. Vector graphics - kind of

The vector graphics output only affects the non-3D rendered elements such as texts, cube axes etc. Normal or glyph plots which are 3D rendered are not affected. Therefore, this option is currently no use for the documentation. It is proposed to create high-quality png-plot with the *Save Screenshot* function and use *GIMP*, *Inkscape*, *convert*

utility from *ImageMagick* or any other tool to convert the pixel to a non-scalable vector graphics copy.

5.7.3. Animations

ParaView allows the creation of animations for your currently selected view entity. So choose your plot coloring and vector entities before creating an animation.

To save an animation from *ParaView* perform the following steps:

1. Click *File* in the menubar
2. Click *Save Animation*
3. Set animation preferences:
   - Animation duration: -
   - Frame rate: $\geq$15
     The human brain perceives successive images as moving, but not necessarily smooth, scene from about 14 to 16 frames per second.
   - No. of Frames/timestep: 1
   - Number of Frames: -
   - Resolution: higher value $\geq$ 1000
   - Timestep Range: Start- and end time step of interest
   - Stereo Mode: *No Stereo*
   - Compression: Checked
4. Click *Save Animation*
5. Specify the path and *File name*
6. Choose the file type of your liking, either video or multiple images
7. Click *Ok*

For the sake of reproducibility of the created animation save the *ParaView* state as described in section 5.7.4.

The target must be to have the animation and the *ParaView* state file available at all time:

```
animation_name.avi
animation_name.pvsm
```

5.7.4. Save *ParaView* states for exported items

For the sake of reproducibility of the created picture or animation you can save the *ParaView* state. If you want to reproduce the picture or animation you can just load the state and all preferences will be set to the exact values of the saved state.

To save a state perform the following steps:

1. Click *File* in the menubar
2. Click *Save State*
3. Specify the path of the figure or animation directory and the *File name* identical to the figure file name
4. Choose *ParaView state file (*.pvsm)* as file type
5. Click *Ok*

A state can be loaded accordingly:

1. Click *File* in the menubar
2. Click *Load State*
3. Select the state file
4. Click *Ok*

5.8. Preferences

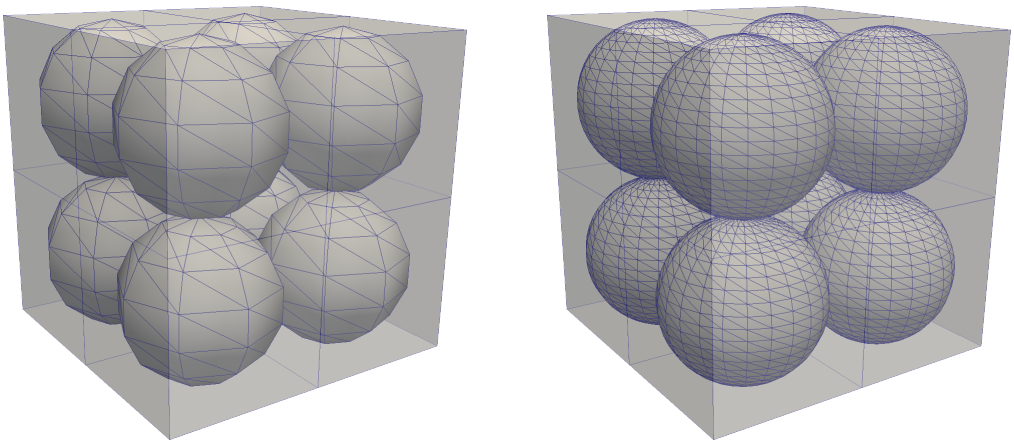
5.8.1. Change mesh on sphere `Glyph`

Sometimes for nice publication pictures the default sphere glyph representation might be a little too angular. Internally, sphere glyphs are nothing but a combination of a certain number of tetrahedron elements. Thus, the mesh behind a sphere glyph might be too coarse. It is assumed you already have a glyph as spheres, otherwise have a look at subsection 5.4.4.

In order to change the mesh size and smooth the sphere shape perform the following steps:

1. Select the Glyph in the *Pipeline Browser*
2. In the *Properties* tab
   - Below the top line press the gear symbol
   - Additional options in section *Glyph Source* appear
   - Adjust the values for *Theta resolution* and *Phi resolution* to your needs. The default coarse value is 8.
   - Click *Apply*

The result is compared in Figure 5.8. The mesh refinement has a significant impact on the computational performance. So only use the refined mesh if it is really necessary.



(a) Resolution=8 (b) Resolution=24

Figure 5.8.: *Peridigm* collocation points inside the base finite element mesh with different glyph resolutions

5.9. Calculate

5.9.1. Cell center

1. Import your model and create a selection if only specific cell volumes are of interest.
2. Select the model or selection in the *Pipeline Browser*
3. From the menu bar:
   - Click Filters
   - Click Alphabetical
   - Click *Cell centers*
4. In the *Properties* tab
   - Select all blocks of interest in the *Composite Data Set Index*
5. Click *Apply*
6. Open a new *Spreadsheet View*
7. In the *Showing* combobox select your *Cell Center Filter Name* and *Point Data* in the *Attribute* combobox
8. Column *points* shows the cell center

5.9.2. Cell volume

A method is described that allows the calculation/extraction of the cell or element volume. This solution is copied from the Paraview Mailing List.

1. Import your model and create a selection if only specific cell volumes are of interest.
2. Select the model or selection in the *Pipeline Browser*
3. From the menu bar:
   - Click Filters
   - Click Data Analysis
   - Click *Programmable Filter*
4. In the *Properties* tab
   - Add to the *Script* field:

```
from vtk.numpy_interface import algorithms as algs

volume = algs.volume(inputs[0])
output.CellData.append(volume, 'volume')
```

   - Select all blocks of interest in the *Composite Data Set Index*
5. Click *Apply*
6. Open a new *Spreadsheet View*

7. In the *Showing* combobox select your *Programmable Filter* and *Cell Data* in the *Attribute* combobox
8. Column *volume* shows the cell volume

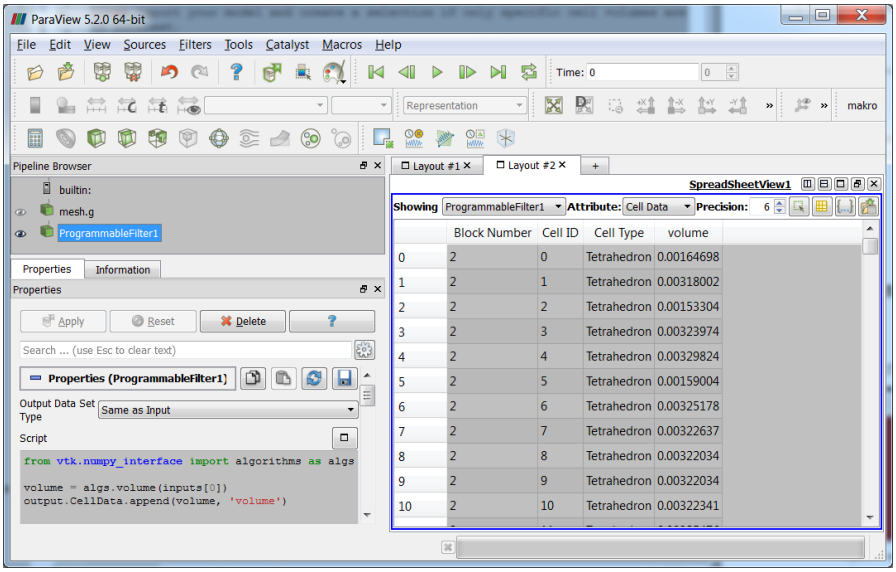


Figure 5.9.: Display of cell volume calculation

6. Benchmark results with *Peridigm*

Peridigm comes with several example problems as well as unit-test models which can be found in the following directories of the *Peridigm* installation folder.

- `/examples/`
- `/test/`

Further examples are available and documented in the *Peridigm* Models and Verification Guide which is part of the same repository.

Bibliography

[1] Martin Rädel and Christian Willberg. *PeriDoX*. GitHub repository. Mar. 2018. DOI: 10.5281/zenodo.1403015. URL: https://github.com/PeriDoX/PeriDoX.

[2] Michael L. Parks et al. *Peridigm Users Guide v1.0.0*. Tech. Report SAND2012-7800. Sandia report. Albuquerque, New Mexico 87185 and Livermore, California 94550, USA, 2012. URL: http://www.sandia.gov/~djlittl/docs/PeridigmV1.0.0.pdf.

[3] Martin Rädel and Christian Willberg. *Peridigm Users Guide*. DLR-IB-FA-BS-2018-23. DLR Report. 2018.

[4] Steve Plimpton et al. *LAMMPS Users Manual*. Albuquerque, New Mexico 87185, 2017. URL: http://lammps.sandia.gov/doc/Manual.pdf.

[5] SIERRA Solid Mechanics Team. *Sierra/SolidMechanics 4.22 User's Guide*. SAND2011-7597. Sandia report. Albuquerque, New Mexico 87185, 2011. URL: http://prod.sandia.gov/techlib/access-control.cgi/2011/117597.pdf.

[6] Florin Bobaru et al. *Handbook of Peridynamic Modeling*. Advances in Applied Mathematics. CRC Press, 2016. ISBN: 978-1315355948.

[7] Stewart A. Silling and E. Askari. "A meshfree method based on the peridynamic model of solid mechanics". In: *Computers and Structures* 83.17-18 (2005), pp. 1526–1535. ISSN: 00457949. DOI: 10.1016/j.compstruc.2004.11.026.

[8] Erdogan Madenci and Erkan Oterkus. *Peridynamic Theory and Its Applications*. Vol. 1. Springer, New York, 2014. ISBN: 978-1-4614-8464-6. DOI: 10.1007/978-1-4614-8465-3. eprint: arXiv:1011.1669v3.

[9] Dan Huang, Guangda Lu, and Pizhong Qiao. "An improved peridynamic approach for quasi-static elastic deformation and brittle fracture analysis". In: *International Journal of Mechanical Sciences* 94-95 (2015), pp. 111–122. ISSN: 0020-7403. DOI: 10.1016/j.ijmecsci.2015.02.018. URL: http://www.sciencedirect.com/science/article/pii/S0020740315000727.

[10] Erdogan Madenci and Selda Oterkus. "Ordinary state-based peridynamics for plastic deformation according to von Mises yield criteria with isotropic hardening". In: *Journal of the Mechanics and Physics of Solids* 86 (2016), pp. 192–219. ISSN: 0022-5096. DOI: 10.1016/j.jmps.2015.09.016. URL: http://www.sciencedirect.com/science/article/pii/S0022509615300909.

[11] G. D. Sjaardema. *APREPRO: An Algebraic Preprocessor for Parameterizing Finite Element Analyses*. Sandia Report SAND92-2291. Sandia National Laboratories Albuquerque, New Mexico 87185-1322 USA, 2010. URL: https://cubit.sandia.gov/public/15.2/help%5C_manual/WebHelp/appendix/aprepro/aprepro.pdf.

[12] Larry A. Schoof and Victor R. Yarberry. *Exodus II: A Finite Element Data Model*. SAND92-2137, UC-705. Sandia report. Albuquerque, New Mexico 87185 and Livermore, California 94550, USA, 1994.

[13] Pablo Seleson and Michael L. Parks. "On the Role of the Influence Function in the Peridynamic Theory". In: *International Journal for Multiscale Computational Engineering* 9.6 (2011), pp. 689–706. ISSN: 1543-1649. DOI: 10.1615/IntJMultCompEng.2011002527. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.226.1934&rep=rep1&type=pdf.

[14] M. R. Tupek, J. J. Rimoli, and R. Radovitzky. "An approach for incorporating classical continuum damage models in state-based peridynamics". In: *Computer Methods in Applied Mechanics and Engineering* 263 (2013), pp. 20–26. DOI: 10.1016/j.cma.2013.04.012. URL: https://dspace.mit.edu/handle/1721.1/108095.

[15] Pablo Seleson. "Improved one-point quadrature algorithms for two-dimensional peridynamic models based on analytical calculations". In: *Computer Methods in Applied Mechanics and Engineering* 282.Supplement C (2014), pp. 184–217. ISSN: 0045-7825. DOI: 10.1016/j.cma.2014.06.016. URL: http://www.sciencedirect.com/science/article/pii/S0045782514002023.

[16] David Littlewood. *Progress and Challenges in Computational Peridynamics*. Workshop on Nonlocal Models in Mathematics, Computation, Science, and Engineering, SAND2015-9303C. Oct. 2015. URL: https://www.osti.gov/scitech/servlets/purl/1332659.

[17] Jesse Thomas, David J. Littlewood, and Timothy Shelton. "Estimation of the Critical Time Step for Peridynamic Models". In: *12th U.S. National Congress on Computational Mechanics, 22-25 July 2013, Raleigh, North Carolina, USA*. SAND2013-5738C. Albuquerque, New Mexico 87185 and Livermore, California 94550, USA, 2013. URL: https://www.osti.gov/scitech/servlets/purl/1140383.

[18] Bahattin Kilic and Erdogan Madenci. "Prediction of crack paths in a quenched glass plate by using peridynamic theory". In: *International Journal of Fracture* 156.2 (2009), pp. 165–177. ISSN: 1573-2673. DOI: 10.1007/s10704-009-9355-2. URL: https://doi.org/10.1007/s10704-009-9355-2.

[19] Stewart A. Silling et al. "Peridynamic States and Constitutive Modeling". In: *Journal of Elasticity* 88 (2007), pp. 151–184. DOI: 10.1007/s10659-007-9125-1.

[20] Daniele Dipasquale et al. "A discussion on failure criteria for ordinary state-based Peridynamics". In: *Engineering Fracture Mechanics* 186 (2017), pp. 378–398. ISSN: 0013-7944. DOI: 10.1016/j.engfracmech.2017.10.011. URL: http://www.sciencedirect.com/science/article/pii/S0013794417303478?via%3Dihub.

[21] Stewart A. Silling. "Reformulation of elasticity theory for discontinuities and long-range forces". In: *Journal of the Mechanics and Physics of Solids* 48.1 (2000), pp. 175–209. ISSN: 00225096. DOI: 10.1016/S0022-5096(99)00029-0.

[22] John A. Mitchell, Stewart A. Silling, and David J. Littlewood. "A Position-Aware Linear Solid Constitutive Model for Peridynamics". In: *Mechanics of Materials and Structures* 10.5 (2015), pp. 539–557. ISSN: 00457949. DOI: 10.2140/jomms.2015.10.539.

[23] John A. Mitchell. *A Nonlocal, Ordinary, State-Based Plasticity Model for Peridynamics*. Sandia Report SAND2011-3166. Multiphysics Simulation Technologies, Sandia National Laboratories, 2011. URL: http://prod.sandia.gov/techlib/access-control.cgi/2011/113166.pdf.

[24] Christopher J. Lammi and Tracy J. Vogler. *A Nonlocal Peridynamic Plasticity Model for the Dynamic Flow and Fracture of Concrete*. Sandia Report SAND2014-18257. Sandia National Laboratories, 2014. URL: http://prod.sandia.gov/techlib/access-control.cgi/2014/1418257.pdf.

[25] J.T. Foster et al. "Implicit time integration of an ordinary state-based peridynamic plasticity model with isotropic hardening". In: *ASME International Mechanical Engineering Congress and Exposition, Houston, Texas, November 9-15, 2012*. 2012.

[26] John A. Mitchell. *A Nonlocal, Ordinary, State-Based Viscoelasticity Model for Peridynamics*. Sandia Report SAND2011-8064. Multiphysics Simulation Technologies, Sandia National Laboratories, 2011. URL: https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/SAND2011-Viscoelasticity.pdf.

[27] John T. Foster, Stewart A. Silling, and Weinong W. Chen. "Viscoplasticity using peridynamics". In: *International Journal for Numerical Methods in Engineering* 81.10 (2010), pp. 1242–1258. ISSN: 1097-0207. DOI: 10.1002/nme.2725. URL: http://dx.doi.org/10.1002/nme.2725.

[28] John T. Foster, Stewart A. Silling, and Weinong Chen. "An Energy based Failure Criterion for use with Peridynamic States". In: *International Journal for Multiscale Computational Engineering* 9.6 (2011), pp. 675–688. ISSN: 0020-7683. DOI: 10.1615/IntJMultCompEng.2011002407. URL: http://www.dl.begellhouse.com/journals/61fd1b191cf7e96f,5dbda3444284ba35,7d22e9255b968290.html.

[29] Christian Willberg, Lasse Wiedemann, and Martin Rädel. "A mode-dependent energy-based damage model for peridynamics and its implementation". In: *Journal of Mechanics of Materials and Structures* (2018).

[30] Stewart A. Silling and R. B. Lehoucq. "Peridynamic Theory of Solid Mechanics". In: *Advances in Applied Mechanics*. Ed. by Hassan Aref and Erik van der Giessen. Vol. 44. Advances in Applied Mechanics. Elsevier, 2010, pp. 73–168. ISBN: 9780123808783. DOI: 10.1016/S0065-2156(10)44002-8. URL: http://www.sciencedirect.com/science/article/pii/S0065215610440028.

[31] David Littlewood and Tracy Vogler. *Modeling Dynamic Fracture with Peridynamics, Finite Element Modeling, and Contact*. Albuquerque, New Mexico 87185 and Livermore, California 94550, USA, 2011. URL: http://www.sandia.gov/~djlittl/docs/Littlewood_USNCCM2011.pdf.

[32] David J. Littlewood et al. "The Peridigm Framework for Peridynamic Simulations". In: *12th U.S. National Congress on Computational Mechanics, 22-25 July 2013, Raleigh, North Carolina, USA*. SAND2013-5927C. Albuquerque, New Mexico 87185 and Livermore, California 94550, USA, 2013. URL: https://cfwebprod.sandia.gov/cfdocs/CompResearch/docs/Littlewood%5C_USNCCM2013.pdf.

[33] W. J. Schroeder et al. *VTK File Formats for VTK Version 4.2*. 2003. URL: http://www.math.u-bordeaux1.fr/~mleguebe/docs/file-formats.pdf.

Peridigm keyword index

Appendices

A. This document

A.1. Repository

This document is part of the PeriDoX repository. The complete repository can be found at:

https://github.com/PeriDoX/PeriDoX

A.2. Typesetting

This document was originally typeset using the documentclass `dlrreprt` from the DLR-internal RM-LaTeX package.

The RM-LaTeX package is not publicly available. Therefore, this document is compatible with a bootstrap-version of the documentclass, called `bootstrap_dlrreprt`. `bootstrap_dlrreprt` class is part of this repository.

The compilation is performed with `pdflatex` with the following options:

```
pdflatex --shell-escape -synctex=1 -interaction=nonstopmode %source --
    extra-mem-top=60000000
```

The bibliography is compiled with `biber`. The glossary must be compiled with makeindex or, for Windows, the included batch-script may be used. The keyword index is created automatically.

The general compilation order is:

`pdflatex` → `biber` → `makeindex` → `pdflatex` → `pdflatex`

B. FAQ

B.1. Peridigm

When I call *Peridigm* with `mpirun` I get an error that the program is unable to find the mesh file. What can I do?

- This problem occurs for *CUBIT* mesh files (*.g)
- Before using them in a parallel job you have to decompose the mesh according to the number of processors to be used
- Please consult section 4.1.1.1

Everything works fine but when I call *decomp* I get a `***HDF5 library version mismatched error***` error. Why?

- Check this same question in the *Peridigm* Installation Guide from this same repository.

I get the following error message combination: `Throw test that evaluated to true: !boost::math::isfinite((*force)[i])` and `**** NaN returned by force evaluation.`. Why?

- It may be that your model became numerically unstable. Try to calculate the model again with a decreased value for the solver timestep safety factor.

I get the following error message: `**** Error: getFieldId(), label not found: Force` when using a `Compute Class Parameters`. Why?

- Make sure you request the `Variable` as a normal `Output Variable` as well. Have a look at the remarks in sections 3.8.1.3, 3.8.2.3 & 3.8.3.3.
- E.g. if your `Compute Class Parameter` is a force, `Force` must also be specified in the output section

I get the following error message: `Error! An attempt was made to access parameter "ABC" of type "int" in the parameter (sub)list "ANONYMOUS" using the incorrect type "double"!` in when using the function parser for an input parameter. Why?

- Make sure that the equation for the function parser does not result in a numerical value that can be expressed as an integer. E.g. With the values `#{DISPLACEMENT=1.0}` and `#{VELOCITY=1.0}` the `Final Time` of a solver `Final Time {DISPLACEMENT/VELOCITY}` supposedly is 1.0. However, for the function parser the result is 1 as an integer. As a double value is expected for `Final Time` an error is thrown.

I get the following error message: `Error, the parameter "ABC" is not a list, it is of type "double"!` **using the free format (.peridigm model). Why?**

↗ Make sure there are no tabs in the line of "ABS". If there are, replace them with spaces and adjust the indentation of the line to the free format in your model.

Nothing happens when I use the `Implicit` **solver and time-dependent** `Prescribed Displacements`**. Why?**

↗ This seems to be a bug in the calculation of the forces.

↗ See https://github.com/peridigm/peridigm/issues/23

B.2. ParaView

BSD Documentation License

```
Redistribution and use in source (Docbook format) and 'compiled' forms (
PDF, PostScript, HTML, RTF, etc), with or without modification, are
permitted provided that the following conditions are met:

1. Redistributions of source code (Docbook format) must retain the above
copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in compiled form (transformed to other DTDs, converted
 to PDF, PostScript, HTML, RTF, and other formats) must reproduce the
above copyright notice, this list of conditions and the following
disclaimer in the documentation and/or other materials provided with the
distribution.

3. The name of the author may not be used to endorse or promote products
derived from this documentation without specific prior written permission
.

THIS DOCUMENTATION IS PROVIDED BY THE AUTHOR ``AS IS AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (
INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```